NATIONAL JUNIOR COLLEGE
SENIOR HIGH 1 PROMOTIONAL EXAMINATION
HIGHER 2

COMPUTING 9569

20 September 2023

3 hours

Additional Materials: Electronic version of `TEN.TXT` data file
Electronic version of `HUNDRED.TXT` data file
Electronic version of `THOUSAND.TXT` data file
Electronic version of `PLAYER_STATS.TXT` data file
Electronic version of `GAME.CSV` data file
Electronic version of `RENTAL.CSV` data file
Electronic version of `RENTER.CSV` 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.

The number of marks is given in the brackets [ ] at the end of each question or part question. The total number of marks for this paper is 100.

This document consists of **15** printed pages and **1** blank page.

 [Turn Over

Instruction to candidates:

Your program code and answer for each of Question 1 to 5 should be saved in a single `.ipynb` file using Jupyter Notebook. For example, your program code, output and answer for Question 1 should be saved as:

`QUESTION1_<your name>.ipynb`

Your answer to Question 3(b)(i) should be written down on the issued A4 paper. You should write your name down on this paper.

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

1 Name your Jupyter Notebook as:

`QUESTION1_<your name>.ipynb`

For this question you are provided with three text files, each contains a valid list of positive integers, one per line:

- `TEN.txt` has 10 lines
- `HUNDRED.txt` has 100 lines
- `THOUSAND.txt` has 1000 lines.

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 `task1_1(filename)` that:

- takes a string filename which represents the name of a text file
- reads in the contents of the text file
- returns the content as a list of integers. [3]

Call your function `task1_1` with the file `TEN.txt`, printing the returned list and its length, using the following statements:

```
result = task1_1('TEN.txt')
print(result)
print(len(result))
```

[2]

Task 1.2

One method of sorting is the bubble sort. Write a function `task1_2(list_of_integers)` that:

- takes a list of integers
- implements a bubble sort algorithm
- returns the sorted list of integers. [5]

Call your function `task1_2` with the contents of the file `TEN.txt`, printing the returned list, for example, using the following statement:

```
print(task1_2(task1_1('TEN.txt')))
```

[1]

Task 1.3

Another method of sorting is the quicksort.

Write a function `task1_3(list_of_integers)` that:

- takes a list of integers
- implements a quicksort algorithm
- returns the sorted list of integers. [7]

Call your function `task1_3` with the contents of the file `TEN.txt`, printing the returned list, for example, using the following statement:

```
print(task1_3(task1_1('TEN.txt')))
```

[1]

Question 1.4

In quicksort algorithm a pivot is chosen which divides the data set into two subsets around the pivot.

(a) **(i)** State the ideal pivot for the quicksort algorithm to execute most efficiently. [1]

(ii) State the difficulty in locating this ideal pivot. [1]

Sometimes the item in the first or last position in the data set is used as the pivot. An alternative is to pick the pivot at random.

A given data set is largely sorted.

(b) Explain what advantage random selection has over selecting the item in the first or last position. [2]

(c) Explain why a programmer might choose to use an insertion sort rather than quicksort in this situation. [4]

2 Name your Jupyter Notebook as:

`QUESTION2_<your name>.ipynb`

A game developer is writing code for a system to hasten the loading of graphical assets ingame. In the game, when the player enters a zone called *dungeons*, the graphical assets is loaded into the memory before being rendered. There are 130 distinct dungeons in total and players can teleport from one dungeon to another. The teleportation to a dungeon is termed *entering a dungeon*. The programming language used allows the use of dynamic memory allocation.

The following algorithm describes what is to happen after a player enters a dungeon:

```
When a player enters a dungeon:
    Has the player visited the dungeon before?
        Yes:
            Assets are loaded from the memory
        No:
            Get some memory that will be used to store the
graphical assets
            Store the assets into the memory
            Assets are loaded from the memory
```

After hours of playtesting entering multiple different dungeons, the game stuttered and slowed down. The developer was asked to investigate and finds no obvious problem. The playtester was asked to restart the game and it runs smoothly again. Several hours later the game stuttered and slowed down again.

You may assume that there are no hardware faults and the rest of the game code is working as intended.

(a) Explain how a mistake in the algorithm is causing the recurring problem. [3]

(b) Explain how the algorithm could be changed to prevent the problem recurring. [2]

3 Name your Jupyter Notebook as:

`QUESTION3_<your name>.ipynb`

(a) Explain what is meant by an object in object-oriented programming. [2]

(b) **(i)** A student is writing a program to represent people in a tennis competition. Assistants, stewards and operators are all employed by the competition organiser. Besides these workers, there are external partners such as the cleaners, umpires and ball person, which are not under employment of the organiser. The players themselves are self-employed.

The competition has 2 categories, one for male players and another for female players. The two groups of players have to be tracked separately as they are ranked with a different ranking system. The student's program contains a class with the identifier `Person`. Sub-classes share the characteristics of this class. Draw an inheritance diagram which includes subclasses `Assistant`, `Steward`, `Operator`, `Cleaner`, `Umpire`, `BallPerson`, `Staff`, `NonStaff`, `Player`, `PlayerMan` and `PlayerWoman`. [2]

(ii) Explain why inheritance is an important feature of object-oriented programming. State an example in this context. [3]

It is common for the properties of a class to be private.

(iii) Explain an advantage of using private properties. [2]

4 Name your Jupyter Notebook as:

`QUESTION4_<your name>.ipynb`

A program is to be written to implement a simulation of a tennis point by two players using object-oriented programming (OOP). Each player is given a name and a set of attributes representing their capability, strength, agility and stamina. The base class will be called `Player` and is designed as follows:

Player
name : STRING
strength : INTEGER
agility : INTEGER
stamina : INTEGER
constructor(n : STRING, st : INTEGER, agi: INTEGER, stam: INTEGER)
set_name(s : STRING)
set_st(s : INTEGER)
set_agi(s : INTEGER)
set_stam(s : INTEGER)
get_name()
get_st()
get_agi()
get_stam()

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 4.1

Program code
```

Output:

Task 4.1

Write program code to define the class `Player`. [5]

In tennis, the winner of the match is determined via the winner's ability to beat the opponent in the smallest subdivisions of the match called *points*.

Task 4.2

We will include a `play()` method for the `Player` class to simulate the winner of a play of a point between two players.

Implement a `play()`method for the `Player` class that:

- takes a `Player` object
- compute a Boolean value based on the attributes of the two player objects based on the following procedure.

  Let x_1, y_1 and z_1 be the strength, agility and stamina attributes of player 1. Similarly, x_2, y_2 and z_2 be the strength, agility and stamina attributes of player 2. Furthermore, we define

$$w_1 = 0.25x_1 + 0.35y_1 + 0.4z_1,$$
$$w_2 = 0.25x_2 + 0.35y_2 + 0.4z_2.$$

  With w_1, w_2 we compute the value p,

$$p = \frac{e^{w_1}}{e^{w_1} + e^{w_2}},$$

  where e is the Euler constant, which is approximately 2.718. You may use `math` module's `exp()` method to get the values as well.

  The value p represents the chance of the player which is represented by the object of which the `play()` method is invoked wins the point.

- returns the Boolean value `True` if a randomly generated value between 0 and 1 is less than p. Otherwise, the Boolean value `False` is returned instead. [6]

Task 4.3

The file, `PLAYER_STATS.TXT`, holds details of the 4 players characters.

The format of the data in the file is:

`<NAME>, <STRENGTH>, <AGILITY>, <STAMINA>`

Write a program code that:

- create an empty list
- read the lines from the file
- extract the `<NAME>`, `<STRENGTH>`, `<AGILITY>`, and `<STAMINA>` values
- store each set of values from each line in a `Player` object
- store all the `Player` objects into the list. [5]

Print out the names of the players represented by the `Player` objects using object method.

[1]

Task 4.4

Write a program code that:

- compute the p values as defined in Task 4.2 between all possible pairs of players in the list.
- report the values, up to 3 decimal places, computed in a two dimensional table format as given in the following example.

```
          , <p_1_name>   , <p_2_name>   , <p_3_name>   , <p_4_name>
<p_1_name>, X            , <p_value_1_2>, <p_value_1_3>, <p_value_1_4>
<p_2_name>, <p_value_2_1>, X            , <p_value_2_3>, <p_value_2_4>
<p_3_name>, <p_value_3_1>, <p_value_3_2>, X            , <p_value_3_4>
<p_4_name>, <p_value_4_1>, <p_value_4_2>, <p_value_4_3>, X
```

where:

- `<p_m_name>` represents the name of the player m
- `<p_value_m_n>` represents the p value calculated when player m plays player n

Note that m cannot be the same as n, as such the entry in the table is denoted as `X`.

[4]

Task 4.5

The second smallest subdivision of a match is called a *game.* Once a player accumulates 4 points, the player will win the game the points are being played in.

A unique feature of tennis is that when a game is tied at 3 points for both players, the game enters a *deuce* situation.

In a deuce, the player that wins the next point after a deuce is said to have the *advantage*. If the player with the advantage wins the next point, the player with the advantage wins the game. If the player with the advantage loses the point, the game is again a deuce, and this process is repeated.

We will write a function to simulate an outcome between two players in a deuce situation.

Write a function `deuce_win(player_one, player_two)` that:

- takes two `Player` objects `player_one`, `player_two`
- returns the integer `1` if the simulation ended with the player represented by `player_one` object wins the game

- returns the integer `0` if the simulation ended with the player represented by `player_two` object wins the game [4]

Task 4.6

Write a program code to:

(a) repeat the simulation of the outcomes of a deuce situation between the following players:

`NOLE, 192, 155, 200`

`CARLITOS, 190, 155, 201`

for 500000 times.

(b) print out a float representation of the proportion of games won by `NOLE` out of the 500000 simulations. [5]

5 Name your Jupyter Notebook as:

`QUESTION5_<your name>.ipynb`

An avid video game collector had a huge collection of video games. To supplement his income, he rented out some of them to his friends and people he meets online. He currently keeps paper records about the renters, video games and the video games loaned. As his operation grows larger, he had difficulty tracking all the rentals he had made and he wants to create a suitable database to store the data and to allow him to run searches for specific data. The database will have three tables: a table to store video games information, a table about the renters and a table about the rentals. The fields in each table are:

`Game`:

- `GameID` - unique video game number, for example, 12
- `Title` - the video game title
- `Genre` - the type of game, for example, Action, RPG, Shooter.
- `TimeSinceRelease` - a float representing the fractional year from the time of release of the game to current time.

`Renter`:

- `RenterID` - renter's unique number, for example, 64
- `Name` - renter's name / alias
- `ContactNo` - renter's phone number
- `ContactEmail` - renter's email address.

`Rental`:

- `RentalID` - the loan's unique number, for example, 12
- `RenterID` - renter's unique number,
- `GameID` - the unique video game number
- `DateLoaned` - the date that the game was taken out by the member
- `Returned` - `TRUE` if the game has been returned, or `FALSE` if it has not been returned.

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 5.1

        Program code
```

`Output:`

Task 5.1

Write a Python program that uses SQL code to create the database `LIBRARY` with the three tables given. Define the primary and foreign keys for each table. Name your database `library.db` . [6]

Task 5.2

The text files `GAME.csv`, `RENTER.csv` and `RENTAL.csv` 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. [5]

Task 5.3

Write a Python program that:

- ask for input of a renter's number from the user,
- uses SQL to find the names of all the games that they have rented out, and whether each game has been returned,
- prints out all the found games and their rental status. [6]

Test your program by running the application with the member number 20. [1]

Save your Jupyter Notebook.

Task 5.4

A measurement for popularity of a game, m_{pop}, is the frequency of the rentals of the game divided by the period, in terms of fractional years, the game had been out.

Write a Python program and the necessary files to create a web application, that displays the following data about the popularity of the games in the collection in the descending order of popularity:

- video game ID
- video game title
- the measurement of popularity of the video game, m_{pop}.

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

Save your Python program as:

`TASK_5_4_<your name>.py`

with any additional files / subfolders in a folder named:

`TASK_5_4_<your name>` [9]

Run the web application.

Save the webpage output as:

`TASK_5_4_<your name>.html` [2]

 END