Name:_____ **Class:**_____

JURONG PIONEER JUNIOR COLLEGE

2022 JC2 Mid-Year Examination

COMPUTING **9569/02**
Higher 2 **26 May 2022**

Paper 2 (Lab-based) **3 hours**

Additional materials:
`Task2_Code.txt`
`Task2_Data.txt`
`Task3.py`
`game_server.py`
`school_info.csv`
`subjects_offered.csv`
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** the 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.

You are reminded to 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**.

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

Instructions to candidates:

Your program code and output for each of Task 1 to 2 should be downloaded in a single `.ipynb` file. For example, your program code and output for Task 1 should be downloaded as `TASK1_<your class>_<your name>.ipynb`.

1 An arithmetic expression is stored as string variable `exp` that contains a combination of integer operands, and arithmetic operators `'+'` and `'-'` only. The result of the arithmetic expression is to be evaluated according to the operator precedence, and its result returned as an integer. You may assume all expressions are valid.

Task 1.1
Write the code for a function `calculate_1(exp)` that takes in `exp` as input parameter and returns the integer value of the arithmetic expression that it takes in.

For example:
`calculate_1("123 + 456" )` returns `579`.

[4]

Task 1.2
The use of one pair of parenthesis `'('` and `')'` is extended to the arithmetic expression `exp`. Use and modify your code taken from Task 1.1 to code function `calculate_2(exp)`.

For example:
`calculate_2("123 - 456 + 789")` returns `456`, and
`calculate_2("-123 + 456 + 789")` returns `1122`.

[5]

Task 1.3
Nested parentheses can now be used in the arithmetic expression `exp`. Write the code for a function `calculate(exp)` that evaluates and returns the integer value of its input parameter `exp`. The examples below show the integer values of various arithmetic expressions that are evaluated and returned by the function `calculate`.

`calculate('123 - (456 + 789)')` returns `-1122`,
`calculate('12 - (34 - 56) + 78')` returns `112`,
`calculate('(12 - 34) + (56 - 78)')` returns `-44`, and
`calculate('1 + (2 - (3 + 4) + 5) - 6')` returns `-5`.

[8]

Task 1.4
Test the code for `calculate(s)` using the following expressions:

- `123-456+789`
- `-123+456+789`
- `123-(456+789)`
- `1+(2-(3+4)+5)-6`

[4]

Download your program code and output for Task 1 as
`TASK1_<your class>_<your name>.ipynb`

2 A computing student would like to store strings in a Binary Search Tree (BST).

Task 2.1

The value of a string `s` can be calculated with the following pseudocode:

```
FUNCTION string_value(s: STRING) RETURNS INTEGER
  DECLARE total: INTEGER
  total ← 0
  FOR i ← 0 to LENGTH(s) - 1
    total ← total + ASCII(s[i])
  NEXT i
  RETURN total
ENDFUNCTION
```

Write the code for the function `string_value`. **[4]**

Task 2.2

Use the following constructor methods for `Node` and `BST` classes:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class BST:
    def __init__(self):
        #The BST is empty when initialised
        #Constructor method
        self.root = None
```

Copy the code provided and add the following methods to `BST` class:

i. `insert(s)` that inserts the string `s` into the BST. If `s` has the same value as an existing node, `s` should be discarded.

ii. `in_order_list()` that returns a list of the strings in the BST, using in-order traversal. **[12]**

Task 2.3

Level-order traversal is another form of traversal where the nodes of the BST are visited level by level. The algorithm to perform level-order traversal is described below:

1. Check if the tree is empty. If the tree is empty, display "`Empty Tree!`".
2. Create a Queue.
3. Enqueue the root of the BST into the Queue.
4. Iterate through the Queue while it is not empty and perform the following:
    i. Remove the first node in the Queue and display its data.
    ii. Enqueue the removed node's left child into the Queue.
    iii. Enqueue the removed node's right child into the Queue.

The file `Task2_Code.txt` contains code for the `Queue` class. Copy and paste the provided code into Jupyter Notebook.

Extend your `BST` class with the `level_order_traversal()` method using the algorithm described. **[7]**

Task 2.4
Write code to perform the following:

- Instantiate a BST named `Tree` and insert the strings from `Task2_Data.txt` into `Tree`.
- Display the strings using:
  i. in-order traversal, and
  ii. level-order traversal. **[5]**

Download your program code and output for Task 2 as
`TASK2_<your class>_<your name>.ipynb`

3 Program a simple turn-based 2-player game of Tic-Tac-Toe. The file `Task3.py` contains a simple library that defines some constants and a `TicTacToe` class to handle the game logic. The following is a summary of the methods in `TicTacToe` class:

Methods	Description
`render_row(row_index)`	Returns a string representation of the specified row, such as: `1 | 2 | 3`

`render_board()`	Returns a string representation of the entire board, such as:
	`1 | 2 | 3`
	`-----------`
	`4 | 5 | 6`
	`-----------`
	`7 | 8 | 9`

`make_move(player_index, cell_index)`	Modifies the board such that the specified cell is marked with the symbol for the specified player.
`is_valid_move(cell_index)`	Returns whether the specified cell is currently blank.
`is_full()`	Returns whether the entire board has been filled up.
`get_winner()`	Returns winning player for the current board or `None` if there is no winner.

Task 3.1
Using this library, write the code for a server program that creates a `TicTacToe` object to store information about the Tic-Tac-Toe board.

Download `game_server.py` and fill the missing codes. Save the updated code as `game_server_<your class>_<your name>.py`.

Analysing this server code, we see that communications with the client is divided into several steps that repeat in an infinite loop:

1. Display current Tic-Tac-Toe board.
2. Check if opponent has won, and if so, end game with opponent winning.
3. Check if the board is full, and if so, end game with a message.

4. Prompt for input from player; if player makes a valid move, update game board accordingly, then send `b'MOVE'` followed by the chosen cell number and `b'\n'` to the opponent; if player chooses to quit, send `b'END\n'` to the opponent and end game with the opponent winning.
5. Display current Tic-Tac-Toe board again.
6. Check if player has won, and if so, end game with player winning.
7. Check if the board is full, and if so, end game with a stalemate.
8. Receive opponent's action via the socket; if the action is `b'MOVE'` followed by a cell number and `b'\n'`, update game board accordingly; if the action is `b'END\n'`, end game with the player winning. **[8]**

Task 3.2
As written, the server player always starts first. This means that the client code should start by receiving and processing the server's result. In addition, Tic-Tac-Toe is a symmetrical game (other than the choice of the starting player). Hence, the client code should be similar to the server code, except that "client" and "server" are exchanged and the last step is moved to the front.

Write the code for the client program. Save the file as
`game_client_<your class>_<your name>.py`. **[8]**

Task 3.3
Run the server and client using two different copies of Python on the same machine to verify that the game works as expected. A sample run is provided below:

Save the screenshot of a sample run where the opponent wins as
`TASK3_<your class>_<your name>.jpg`

The following is a sample **server** output:

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9

Server moves (0 to quit): 1

 O | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9

Client moves: 4

 O | 2 | 3
-----------
 X | 5 | 6
-----------
 7 | 8 | 9

Server moves (0 to quit): 2
```

```
 O | O | 3
-----------
 X | 5 | 6
-----------
 7 | 8 | 9

Client moves: 5

 O | O | 3
-----------
 X | X | 6
-----------
 7 | 8 | 9

Server moves (0 to quit): 3

 O | O | O
-----------
 X | X | 6
-----------
 7 | 8 | 9

You win!
```

[1]

The following is a sample **client** output:

```
Server moves: 1

 O | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9

Client moves (0 to quit): 4

 O | 2 | 3
-----------
 X | 5 | 6
-----------
 7 | 8 | 9

Server moves: 2

 O | O | 3
-----------
 X | 5 | 6
-----------
 7 | 8 | 9

Client moves (0 to quit): 5
```

```
Client moves (0 to quit): 5

 O | O | 3
-----------
 X | X | 6
-----------
 7 | 8 | 9

Server moves: 3

 O | O | O
-----------
 X | X | 6
-----------
 7 | 8 | 9

Opponent wins!
```

4 A private institute of higher learning uses flat files to store public data about public schools in Singapore and information about the subjects offered by each school. Schools are located in a zone, and identified by which level of students they teach.

The institution wishes to migrate this information into a database. A web application will then be used to perform queries and display the results.

Task 4.1
Create an SQL file called `TASK4_1_<your class>_<your name>.sql` with the SQL code to create database `schools.db` with two tables: `School` and `Subject`.

The `School` table will have the following fields:

- `SchoolID` – an integer value
- `Name` – the full name of the school
- `Zone` – the zone which the school is located in
- `Level` – the level of students that the school takes in
- `YearsOfStudy` – the number of years a student takes to graduate

The `Subject` table will have the following fields:

- `SchoolID` – an integer value
- `SubjectName` – the name of subject

The primary key of `School` table is the `SchoolID` column, and the composite keys of `Subject` table are `SchoolID` and `SubjectName`.

Save your SQL code as
`TASK4_1_<your class>_<your name>.sql` **[6]**

Task 4.2
Python programming language and object-oriented programming will be used to publish the database content on a web page.

The class `School` will store the following data:

- `id` – stored as an integer
- `name` – stored as a string
- `zone` – stored as a string

The class has the following methods defined on it:

- `safe_name()` – returns a string containing no spaces, which may be safely used for older computer systems. This string should be constructed as follows:
  - name with all spaces replaced with underscore `'_'`
  - apostrophes `''`, periods `'.'` and commas `','` removed
- `as_record()` – returns a tuple of values in the same order as the SQL table column order

The subclasses `PrimarySchool`, `SecondarySchool`, and `JuniorCollege` inherit from `School`, such that:

- they have class attributes with an appropriate value
  - the attribute level has values "`primary`", "`secondary`", or "`junior college`"
  - the `PrimarySchool` subclass has attribute `YearsOfStudy` = 6
  - the `SecondarySchool` subclass has attribute `YearsOfStudy` = 4
  - the `JuniorCollege` subclass has attribute `YearsOfStudy` = 2
- the as_`record()` method returns the appropriate values for the subclass

Save your program code as
`TASK4_2_<your class>_<your name>.py`

[10]

Task 4.3

Write program code to read the records from the flat file `school_info.csv`, creating an instance of the appropriate class for each school. After which, insert all information from the file into the `schools.db` database.

Add your code to
`TASK4_2_<your class>_<your name>.py` **[5]**

Task 4.4

Write program code to read the records from the flat file `subjects_offered.csv`, and insert all information from the file into the `schools.db` database.

Add your code to
`TASK4_2_<your class>_<your name>.py` **[4]**

Task 4.5

The institute wants to filter the schools by the name of school and have the results displayed in a web browser.

Write the code for a Python program as well as the necessary HTML files to create a web application that:

- gets user to enter a (full) school name in a HTML form,
- receives the data from the form,
- perform a query on the database, and
- displays the school's information and subjects offered in a neat HTML table.

Save your Python program as
`TASK4_5_<your class>_<your name>.py` **[8]**

Task 4.6
Run the web application with "`JURONG PIONEER JUNIOR COLLEGE`" entered.

Save the output of the web app as
`TASK4_6_<your class>_<your name>.html` **[1]**

<END OF PAPER>