**NANYANG JUNIOR COLLEGE**

**JC2 COMMON TEST 2**

Higher 2

**COMPUTING 9569/02**

Paper 2 (Lab-based) **4th July 2023**

**3 Hours**

Additional Materials: Removable storage device

Electronic version of SUNSHINE.csv data file

Electronic version of WORDS.txt data file

Electronic version of REQUEST.txt data file

Electronic version of STUDENTS.txt data file

Electronic version of CANDIDATES.txt data file

Electronic version of VOTES.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.

**Instruction to candidates:**

Your program code and output for each of Task 1 and 2 should be saved in a single .ipynb file. For example, your program code and output for Task 1 should be saved as

TASK1\_<your name>\_<class>.ipynb

**1** The text file SUNSHINE.csv contains comma-delimited data on the monthly mean sunshine hours in a day, recorded at the Changi Climate Station in Singapore, for the period 1982 to 2023.

Each line within the file is in the following format:

YYYY-MM, <float>

* the date in the form YYYY-MM, (for example 1982-01 is the month of January in 1982)
* the mean daily sunshine duration per month, in hours with decimals.

The data contained in the text file is presently unsorted.

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**

Insertion sort is an algorithm used for sorting data.

Write a function task1\_1(filename) that:

* takes in the name of a file, filename
* reads in the mean daily sunshine duration data from a text file, as a list, filtering out the year 2023
* implements an insertion sort algorithm
* sorts the data in ascending order of date
* returns the sorted list of data.

Use SUNSHINE.csv to test your program code.

Show your outputs clearly. [9]

**Task 1.2**

Write a function task1\_2(sortedlist) that:

* takes in a list of sorted data sortedlist returned by **Task 1.1**
* calculates and displays **for each year**:
  + the lowest mean daily sunshine duration and the (numerical) month in which it occurs, as a tuple
  + the highest mean daily sunshine duration and the (numerical) month in which it occurs, as a tuple
  + the range of mean daily sunshine duration (the difference between the lowest mean daily sunshine duration and the highest mean daily sunshine duration).

Use SUNSHINE.csv to test your program code.

Show your outputs clearly. [8]

**Task 1.3**

A researcher wants to analyse how the mean daily sunshine duration has changed over the years 1982 to 2022.

Write a function task1\_3(sortedlist) that:

* takes in a list of sorted data sortedlist returned by **Task 1.1**
* outputs the average mean daily sunshine duration of each month (Jan, Feb, ...)  
  for the years 1982 to 2022 (i.e. average of 1982 Jan, 1983 Jan, 1984 Jan, ...).

Your output format should be as follows:

Month 1: average daily sunshine duration 1982-2022 was <float> hours.

Month 2: average daily sunshine duration 1982-2022 was <float> hours.

...

Month 12: average daily sunshine duration 1982-2022 was <float> hours.  
  
Use SUNSHINE.csv to test your program code.

Show your outputs clearly. [5]

(Total: 22)

**2** A binary search tree is used to store words for a venture capital firm.

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: |

## The tree is implemented using Object-Oriented Programming (OOP).

## Task 2.1

Write program code to declare the Node class. Each Node represents a record, with the following attributes:

* left - the left Node
* right - the right Node
* data - the data stored in the Node. [3]

**Task 2.2**

Write program code to declare the BinarySearchTree class, with the following attributes:

* root - the root Node

BinarySearchTree should be initialised without any arguments. It should have the following methods:

* insert(data) encapsulates data in a Node, and inserts it at an appropriate position in the tree by comparing the word key of the node data. If data already exists, output a suitable error message.
* find(word) returns the data from the first Node found with a matching word, or None if not found.
* inorder() returns a list of **sorted data items** from traversing the node’s subtree using in-order traversal.

You can include additional helper methods to the above classes, where needed. [15]

Test the BinarySearchTree by inserting the following data items, each as a Python dict:

1. {'word': 'property', 'definition': 'thing(s) belonging to someone'}
2. {'word': 'obtain', 'definition': 'come into possession of'}
3. {'word': 'venture', 'definition': 'undertake a risky course of action'}
4. {'word': 'policy', 'definition': 'a plan of action adopted by an individual or organisation'}
5. {'word': 'generate', 'definition': 'bring into existence'}
6. {'word': 'yield', 'definition': 'give or supply'}
7. {'word': 'stock', 'definition': 'capital raised by a corporation through the issue of shares'}

The data items can be found in WORDS.txt for convenience.

Using the find() method, output the definition of the word 'venture'.

Display the sorted list of data items returned when the inorder() method is called. [5]

**Task 2.3**

The list of sorted data items was then sent to another programmer, who did not have access to the Binary Search Tree. He decided to implement a Binary Search algorithm to find items within the list.

Write a function task2\_3(sorted\_list, word) to:

* accept sorted\_list and word as parameters:
  + sorted\_list, the sorted list of data items obtained from **Task 2.2**
  + word, to identify the item being searched for
* implement a binary search algorithm to return the data in the sorted\_list that matches the word
* return None if there is no matching data.

Using this function, print the definitions of the following words:

* 'yield'
* 'commission'

Show your outputs clearly. [6]

(Total: 29)

**3** HTTP is a text-based protocol that describes the rules for communication between web clients and servers. A web server handles HTTP requests from clients, and returns an HTTP response. An example of an HTTP request is given in the data file REQUEST.txt.

The first line of the request is the header. The header comprises the method, path, and protocol, separated by a space. The line is terminated with a line break. An example of a GET request for the path / using the HTTP version 1.1 protocol is shown below:

GET / HTTP/1.1

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.

## Task 3.1

Write a function parse(req\_str) that:

* takes one argument, a request string req\_str
* extracts the method, path, and protocol,
* returns the header (method, path, and protocol) as a 3-string tuple.

Test the function with the contents of REQUEST.txt. [3]

A minimal HTTP response comprises:

* a status line (protocol and status code), followed by a single line-break
* a content-length header, followed by two line-breaks
* the response body.

If the request path is correct, a minimal response (status code 200) is as follows:

HTTP/1.1 200

Content-Length: 12

Hello World!

If the request path is incorrect, a minimal response (status code 404) is as follows:

HTTP/1.1 404

Content-Length: 14

Page not found

## Task 3.2

Write a function, response(status\_code, message) that:

* takes in a status\_code integer and message string
* uses HTTP/1.1 as the protocol
* uses the given status\_code (a 3-digit number)
* determines the length (number of characters) of the message
* returns an appropriate HTTP response as a string.

Test the function with the above minimal responses. [4]

A web server handles requests from web clients over a network socket and sends appropriate responses for each request. Each request begins with a new socket connection, which is closed after sending the response.

## Task 3.3

Write a Python web server that uses network sockets to receive requests and return appropriate responses in an infinite loop. You must **NOT** use the flask module for this question.

The web server binds to the IP address 127.0.0.1 on port 65535.

Your web server will:

* send a response with a 404 status code containing the message Not found if the request path is **not** / or /index.html
* otherwise send a response with a 200 status code and the message Hello World!.

Save your program as

TASK3\_<your name>\_<your class>.py. [12]

Run the server program. In a web browser, enter the following addresses, saving the webpage displayed as html for each:

* 127.0.0.1:65535/index.html
  + Save the output as TASK3\_3\_1\_<your name>\_<class>.html
* 127.0.0.1:65535/home.html
  + Save the output as TASK3\_3\_2\_<your name>\_<class>.html [2]

(Total: 21)

**4** To facilitate ease of voting and promote greater cohort engagement in the election process, a school’s Student Council has decided to implement an online voting system. The information is to be stored in a database.

For the Student, the following information is recorded:

* MatricID – a unique matriculation ID string allocated to the student
* Class – class of the student
* IndexNo – index number of the student in the class
* Gender – gender of student, either 'M' or 'F'.

For the Candidate, the following information is recorded:

* CandidateID – a unique ID string allocated to each candidate
* Name – name of the candidate
* Slogan – campaign slogan of the candidate.

For the Vote, the following information is recorded:

* MatricID – the matriculation ID allocated to each student
* CandidateID – an ID allocated to each candidate.

A student is eligible to vote for multiple candidates.

It is assumed that the database is already normalised and the information above is to be stored in three different tables.

**Task 4.1**

Create an sql file called TASK4\_1\_<your name>\_<class>.sql to show the SQL code to create the database voting\_records.db with the three tables given. Define the primary and foreign keys for each table, as needed.

Save your SQL code as

TASK4\_1\_<your name>\_<class>.sql [5]

**Task 4.2**

The files STUDENTS.txt, CANDIDATES.txt and VOTES.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 the program.

Save your program code as

TASK4\_2\_<your name>\_<class>.py [5]

**Task 4.3**

The Student Council would like to implement a program to display the names of all the candidates a student has voted for, based on the student’s matriculation number.

Write a Python program and the necessary files to create a web application, that:

* receives the MatricID string from an HTML form, then
* creates and returns an HTML document that enables the browser to display a table with the following data
  + candidate id
  + candidate name
* sorts the list according to the CandidateID, in descending order.

Save your Python program code as

TASK4\_3\_<your name>\_<class>.py

with any additional files / sub-folders as needed in a folder named

TASK4\_3\_<your name>\_<class> [9]

Run the web application.

Save the output of the program when NYJC-2023003 is entered as the MatricID, as

TASK4\_3\_<your name>\_<class>.html [3]

(Total: 22)

**−−−−− END OF PAPER −−−−−**