CSI3344 Distributed Systems

Assignment 3: Distributed system project & video demo

Objectives from the Unit Outline

- Implement a small distributed system using RPC or RMI.
- Apply algorithms to the solution of complex distributed system problems.
- Discuss the structure and functionality of distribution algorithms for distributed systems.
- Present outcomes of the research and/or development of a distributed system.

General Information

- Assignment 3 (or A3 for short) is the major assessment of the unit. This is a group assignment involving teamwork and individual work. Therefore, it is divided into two parts: the first part is a **group assessment** component of **up to two students**, and the second part is an individual assessment component (i.e., a **video presentation** to be completed by individual team members). Therefore, two batches of submissions are required.
  - The first batch of submission includes submission of the project implementation (including source code/s, and executables if applicable) and the written report. This part is a *group* task so it should be completed by all group members and only submitted by the team leader (that is, the other team member should *not* submit it repeatedly).
  - The second batch of submission consists of one *video link* and an *optional Peer-Review* document. This is an individual task/work so each team member of the group should prepare and submit their own version of the work seperately. The video link is the web link to a video presentation, which should contain two parts: a demonstration of the project completed by the group, and an individual's *reflection* on the project work. You can also include an (optional) *Peer-Review* document within this batch of submission.
- Group formation for A3:
  - Groups for Assignment 3 will be randomly assigned in week 6. If you want to form your own team with a friend/classmate you know, you can do so before week 6 – To do this, go to A3 group sign-up page on Canvas (via "People" section, and click on "A3-Groups" panel) to sign in a group, or you can email your tutor your team members' details so your tutor can help you set up a group.
  - Once a group is formed, the group members will be locked to Canvas. - If you want to change the team members of your group, you must email your tutor to request a change. Note that group membership changes can only be made within the first 8 weeks.
  - In some special circumstances, groups of three students are possible (for example, only one student is left unpaired). If you wanted to form a group of three, that's fine, but your group will need to do some extra work (see more details below). On the other hand, if you really want to do A3 alone, that's fine too (though this is discouraged). In such a case, you must let you tutor know your decision by week 8 and you should not expect a reduction in the workload of the assignment.

Due date: *(See Due date in Assignments section on Canvas)*

Marks: 100 marks (which will then be converted to 50% of unit mark)

Background Information and Assumptions of the Project

The *Open University of Science and Technology (OUST)* is interested in developing a simple ***Honors Enrolment Pre-assessment System*** (HEPaS) that can be used by their graduates and/or third-year students to self-assess if they are eligible for *Honors studies.* To be even more useful, the HEPaS also allows other users to use the system.

HEPaS is a simple *three-tier* distributed system. The system has one client-side application, one or more server-side applications, and one database server which holds all OUST students' course learning records.

- The client (application) is an interface that collects data from the user, performs preliminary pre-processing of the data items, and sends the data collected to the remote application/s at server-side.
- The server-side application uses the data received from the client to authenticate the user. After a successful authentication, the server-side application sends a request to the database server to get the user/student's learning records. It then assesses whether the user meets *Honors* enrolment criteria therefore determining if the user is eligible for honors studies. The assessment result is then sent back to the client application.
- After the client application receives the data from the server, it finally displays the assessment results to the user.

In this project, you are requested to design and implement a version of the above-described HEPaS under some practical requirements.

1. The mini project

HEPaS behavior and requirements:

(1). Only authenticated users are allowed to use this application (from the client-side).

(2). The client application allows a user to enter data for the Honors enrolment pre-assessment (HEPa). A user can be a (former or current) student at OUST, or anyone interested in enrolling an Honors course at OUST.
The client application first prompts the user to answer whether he/she is an OUST student. If the user is an OUST student (former or current), the client will ask the user to enter his/her Person ID (or Student ID, which can be e.g., an 8-digit number) and other information (e.g., first and last name, his/her OUST email address, etc.) to authenticate the user. The client then passes the data items entered to the server-side application/s to proceed the HEPa evaluation/assessment.
If the user is not an OUST student, the client requests the user to enter a Person ID and a sequence of unit scores (or marks e.g., in <unit_code, mark> pair), one by one, through keyboard. The number of unit scores entered should be between 16 and 30, including "Fail" (e.g., a unit score below 50) marks and duplicate unit marks (if the student did the same unit multiple times).

(Note: you may assume that the unit_code is a string of up-to 7 characters, like "CSI3344" or "unit_03", etc., and the mark is a real number between 0.0 and 100.0 inclusive, like 82.4 or 26.0 etc.).

The client then passes all the collected data to the server for HEPa assessment. It then waits for the assessment results from the server, and finally displays the results on the screen.

(3). The server application (also known as *server-1*) provides services/operations that process the data received from clients, calculate the averages, evaluate the eligibility based on evaluation criteria, and so on. The evaluation results are then returned to the client. If necessary, the server can send a service request to the database server, such as obtaining an OUST student's course learning records, etc.

The basic operations on the server-1 include (but are not limited to):

- displaying individual scores (in <unit_code, mark> pairs) on the screen in their input order;
- calculating the *course average*;
- selecting the best (or highest) 8 scores, and calculating the *average* of the best 8 scores;
- evaluating the eligibility according to the Honors evaluation criteria (see below); and finally,
- sending the evaluation result back to the client.

The *Honors evaluation/assessment criteria* are as follows (in sequence):

- If the student completed 15 or a smaller number of units, return a message:

  "<Person ID>, <course average>, completed less than 16 units! DOES NOT QUALIFY FOR HONORS STUDY!"

- If there are 6 or more "Fail" results, return a message:

  "<Person ID>, <course average>, with 6 or more Fails! DOES NOT QUALIFY FOR HONORS STUDY!"

- If the course average is greater than or equal to 70, return a message

  "<Person ID>, <course average>, QUALIFIES FOR HONOURS STUDY!";

- If the course average is less than 70 and greater than or equal to 65, but the average of the best 8 scores is greater than or equal to 80, return a message:

  "<Person ID>, <course average>, <best 8 average>, QUALIFIES FOR HONOURS STUDY!";

- If the course average is less than 70 and greater than or equal to 65, and the average of the best 8 scores is less than 80 , return a message:

  "<Person ID>, <course average>, <best 8 average>, MAY HAVE GOOD CHANCE! Need further assessment!".

- If the course average is less than 65 and greater than or equal to 60, but the average of best 8 scores is 80 or higher, return a message:

```
            "<Person ID>, <course average>, <best 8 average>, MAY HAVE A
             CHANCE! Must be carefully reassessed and get the coordinator's
             permission!".
```

- Otherwise, return a message:

```
          "<Person ID>, <course average>, DOES NOT QUALIFY FOR HONORS STUDY!"
```

(4). Database server (also known as *server-2*): it stores (former or current) OUST students' course learning records (OSCLR), which may include student's unit selection/learning history, related unit results, etc. For simplicity, let's assume that server-2 only stores unit results for one course per user and that each student's record can hold unit results for up to 30 units.
The database is sometimes called an *OSCLR* database.
The student learning records in *OSCLR* database are read-only. HEPaS client cannot make any changes to student's learning records in the *OSCLR* database.

(5). Client and server applications should be able to run on different computers.

(6). The client application should do necessary data validations. For example, OUST doesn't allow the students do any unit more than three times. This means that while a student may have up to three scores for the same unit, there must be no more than two "Fail" grades and no more than one "Pass" (i.e., the unit mark is greater than or equal to 50) grade, etc.

Recommended procedure for coding:

You should complete the project in two phases. The first phase is a simple *"two-tier" interaction* between the client application and the server-1 application/s. The second phase is to upgrade the *"two-tier" interaction* to a *"three-tier" interaction* by adding server-2, which acts as a database server (or data center). The database server can handle some requests from the server-1 application/s.

Phase 1: the *"two-tier" version implementation* (team work, *35 marks*):

Refer to the attached Python-like pseudocode in Appendix A that defines a class *Average*. It is an application that runs on a local machine. It accepts a number of integers as scores to some units. Then it adds them up, and finally returns the average score (as a real number). The result is then displayed on the screen.

This application code can be converted to a client-server interaction. For example, the client application (or process) accepts integers (of unit scores) from a process (on local machine), and then sends the data to the server, requesting a server-side application (or process) to calculate the *Average* value, which was then sent back to the client. The result is then displayed in the interface at the client side.

Based on this idea, practice using RMI to complete a mini programming project for a two-tier client-server application for HEPaS. That is, in this phase, it is assumed that the HEPaS users are non-OUST students, therefore it is not necessary to consider any functionalities listed in item (4) in HEPaS behavior and requirements section (see above).

You may start by modifying (or converting) the above-mentioned pseudocode to a simple client-server application to implement the *HEPaS*. That is, the client makes RMI to a remote application/method, say, *Evaluator* (located in the server-1 side), which evaluates a user's eligibility for an Honors study based on the data entered/received.

Recommendations & restrictions to Phase-1 programming:

You are encouraged to choose from one of the following two implementation styles:

(i) Use traditional *synchronous communication* pattern to implement the system (i.e., using RMI or RPC technique/s). While it is OK to use the class library available in the language environment to implement the two- (and/or three-tier) application, no third-party package/s (outsides the language environment) can be used.
In this case, the corresponding third-tier database server (to be implemented in Phase-2) could be simplified to using a set of arrays to mimic a relational database.

(ii) Use *asynchronous communication* pattern (or a combination of *synchronous* and *asynchronous communication* patterns) to implement the system. In this case, in addition to use RMI (or RPC) technique, you can use all class libraries from the chosen programming environment and any third-party package/s to implement the application.
In this case, the corresponding third-tier database server (to be implemented in Phase-2) must use (or be connected to) a real/existing database server such as SQL server, MySQL, or Oracle server, etc.

Remember the steps to create an RMI application:
1. define the interface for the remote object. 2. implement this remote interface. 3. create stub and/or skeleton. 4. implement the server software. 5. implement the client software.

Phase 2: The *"three-tier" version implementation* (team work, *20 marks*):

This phase should be started after you completed the "*two-tier*" client-server application in Phase 1. This phase consists of two sub-tasks:

(1) Extensively test the client/server application (using various data) completed in *Phase 1*.

(2) Create a *third-tier* server and expand the system completed in Phase 1 to a three-tier client-server application for HEPaS:
If a user is a (former or current) OUST student, he/she may request to use his own bachelor's course study historic data stored in the *OSCLR* database to do the Honors study pre-assessment, rather than entering his/her unit-related data via the interface at the client interface.
In this phase, you are requested to create the *third-tier* server to act as the *OSCLR* database (or data center). This database stores students' course learning historic data, such as the students' information, his/her course information, unit selection/learning history, unit results, etc. A student's unit selection/learning history is a list of up to 30 <unit_code, score> pairs (please note that for each unit, while a student may attempt the unit multiple times, only *up-to* one "Pass" grade can be recorded). You should also make necessary extensions to the client and server-1 applications that were implemented in Phase-1 to complete the three-tier system.

For testing purpose, you can manually create some data and enter it into the OSCLR database (as a sample data set you can use the data sets given in the attached documents titled "sampleDataSets.xlsx" – Please note that this file contains three tables: a Student_Info table, a Course_Info table, and a Student_Unit table. Some tables have unpopulated columns – you may fill in some data to make it complete, if needed).

The server-1 and server-2 can communicate by RMI (or PRC). However, the client application has no direct access to the server-2.

- Extension to the *client*-side applications:

Client application is updated to allow a user to do the honors study pre-assessment by one of the two ways:

(i). The user enters his Person ID and a sequence of unit results to do the honors study pre-assessment (i.e., same as what was done in Phase 1); or

(ii). The user uses his/her student records to do the honors study pre-assessment. In this case, the user should enter a Person or Student ID (and other related information for authentication). The client submits a request to the server-1, requesting the server-1 to use the user's unit results stored in the *OSCLR* database (i.e., server-2) for the HEPa evaluation/assessment.

- Extension to the server-side applications:

(iii). Server-1 extension:
When a user wanted to use his/her OUST student course learning records to do the honors study pre-assessment, Server-1 makes a request to server-2 (with

parameters such as the Person/student ID, etc.), asking for the student's course learning records. Once received data from server-2, it extracts related data items to do HEPa evaluation and then returns the evaluation result back to the client. Necessary data validation should be done at server-1 side, if applicable.

Extra work for groups with three students:

(Note: groups of two or fewer students should skip this section)

If your group has three team members, your team is required to do some extra work. Please see ***Appendix B*** for details on the additional work and requirements for your group.

Bonus Marks (up to 5 marks):

Additional work and effort to improve the project and make it more useful can be worth bonus marks. Up to 5 marks can be added for some significant improvements to the HEPaS system.

After you successfully completed Phase 2, if you have time, you are encouraged to do some improvements to earn bonus marks. Some possible improvement work is listed in Appendix B. You may choose to do one or two of the tasks listed there to get up to 5 bonus marks that could be added to the score of your group works (Note: The bonus mark only applies to the case where the project is completed by a group of two students or by individuals. Groups with three students will not receive bonus mark because the tasks are a part of their team assignment).

Note that *Bonus marks* are not required and will not improve your score above 100% of the assignment score.

Other basic requirements

- The application/system should include all required components.
- The observable behaviors of your application should be consistent with what is described/required.
- The application provides necessary error handling mechanisms.
- The application must be implemented using Python – if you got permission to use a different programming language, please use ONLY one single programming language, chosen from a list of o-o programming language set (e.g., Java, C#, C++). However, a SQL-compatible language can be used to implement the third-tier server application/s in Phase-2.
- The system must be runnable *off-the-shell*. That is, executable codes should be produced, and can be run in OS shell (e.g., Windows' Commend Line prompt) or by double clicking on the *executable* in a folder. In other words, the final product of the project (i.e., the system developed) should be runnable independent of any programming environment.
- The project report should be well structured, informative, and not contain codes of your project. All code files should be included, separately, in the submitted .zip file.

2. The written report (team work, 14 marks)

Refer to the **Format requirement of the Assignment report** in page 10. The *main body* should include (but not be limited to):

i. (A brief) Introduction to RMI.
ii. Application requirements.
iii. Application design and implementation procedure.
iv. *User manual*: application set-up and usage steps (with possible screenshots)
(Note: this is to allow your tutor to install your project code/s to their/lab computer/s for testing your project/codes).
v. Test cases that can be used to test your system.
vi. Example snapshots demonstrating application running status (with input/outputs).

3. The video demonstration and reflection

This is an individual task (not a group task). Each team member of the group should prepare and submit their own video version.
The video should cover two parts: the first part is a demonstration of the project completed by the group, and the second part is an individual's *reflection* on the project work. The entire length of the video should be limited to 15 minutes.

(a) The video demonstration/presentation (individual work, 20 marks)

This first part of the video should mainly be a demonstration of the group project, although you may also present other contents such as a brief introduction to RMI and explanation of the project report, etc. Some requirements are:

(1) This first part of the video should be between 5 and 10 minutes in length.
(2) Students Identity Verification (SIV): For academic integrity, a SIV is required for the video demonstration. – At the beginning of the video, you should take a few seconds to briefly introduce yourself while the computer camera captures your face or student ID.
(3) The demo should include
- a brief introduction of software used to develop the project;
- the architecture of the system completed;
- user authentication (and registration if applicable);
- the demonstration over some test cases:
(i) case/s for a user who is a non-OUST student: For example, a user enters his/her Person ID, and a series of valid unit related data items for Honors Study pre-assessment, where each data item is in a format of <unit_code, score> pair;
(ii) case/s for a user who is an OUST student: For example, a user enters a Person ID, OUST email and any other information to do Honors Study pre-assessment using his/her course learning records/data stored in the database at server-2;
(iii) cases with some invalid data input: For example, less than 16 or more than 30 unit marks (or data items) are entered, non-authenticated user trying to use the system, a unit score entered is less than 0.0 or greater than 100.0, etc. - All these should trigger the system to display error handling message/s.

(4) Any special features that you want to show.

(Note: the purpose of a project demonstration is to show potential users of the system about how to use the system, so it should not involve too many system implementation details. Specifically, the implementation code does not need to be explained. During the demo, the system should be run to only display the system behavior (e.g., for a given set of inputs, one can see how the system reacts and what outputs are generated in response to user input, etc.).

(b) The video reflection (individual work, 10 marks)

This is the second part of the video. The length of this (second) video part should be limited to 5 minutes.
In this video part you should explain your own contribution to the teamwork and provide a personal comment about the teamwork. Here is a list of points you can consider in your reflection:

- What was your project about?
- How successful was your team?
- What was your key role in completing this project?
- How did completion of this task relate to your work in this unit or other units you have studied?
- What was your key takeaway from the teamwork/task?
- What difficulties did you (or your team) encounter during the project development and how did you overcome them?
- What (lessons) did you learn from the process, if any?
- Which parts of the tasks, if any, were the most challenging?
- Which parts of the project are done well, and which parts could be improved if given the opportunity?

(As a closing note, if you wish, you can tell a story that you would like to share with your classmates, whether successful or unsuccessful - for example, any tasks/techniques that you practiced on this project that you are proud of, or a problem or technical issue/s that you tried to fix without success, etc.)

This (second) video part may be used to support un-even distribution of marks/scores between team members if there are significantly differences in their project contributions.

(Note: If you wish to split the video in two short video clips, e.g., one for the project demo and the other for personal reflection, you can. In this case, please submit two separate video links instead of one)

4. The peer review (*optional, individual work - 0 mark*)

This is to grade performance for each team member, including yourself.

This task is *optional* – If you choose to do it, please download the *peer review* form template from Canvas, fill out the form with necessary remarks/comments, and finally submit it.

Format requirement of the *written report*

Report content	**Cover/title page** Must show unit code, unit title, assignment title, assignment due date, student IDs and name/s of all team members, etc.
	Executive Summary This should represent a snapshot of the entire report that your tutor will browse through and should contain the most vital information requested. This part should be placed in between Cover page and ToC.
	Table of Contents (ToC) This must accurately reflect the content of your assignment/report and should be generated automatically in Microsoft Word with clear page numbers.
	Introduction Provide background information on the project, defining its scope and assumptions if any; Use in-text citations/references where appropriate. **Main body** (this should be divided into sections, each with appropriate titles). This part should contain (but not limited to): • Understanding of concepts/techniques involved in the report. • The problems being solved. • Strategies used to solve the problem (e.g., an approach/es for developing solution/s, etc.). • *User Manual* for installing your project onto (two or more) computers and running your programs. • Test cases that your team members may use to demonstrate the project in your video demo/presentation. • Comments/discussions or criticisms of the solutions developed /implemented, etc. • Anything else you feel is necessary to include. **Conclusion** Major achievements or findings of the project/assignment. **References** A list of end-text references, formatted according to the ECU requirements using the APA format (Web references are OK)
Other requirements	The length of the assignment/report should be generally of 3000~4000 words (excluding references, screen shots and diagrams/charts). The text must use font **Times New Roman** and **be not smaller than 12pt**.

Submission requirements (1 mark)

Project submission (one submission per team):
This submission should include the written report, all implementation codes (source codes and executable files, if applicable), and any additional documentation related to the report/project (if any). This submission can only be submitted by the team leader (i.e., through the team leader's submission portal). Other team member/s should not submit duplicate version of the assignment.

Submit only the final project product (that is, only submit the 3-tiered system, do not submit the 2-tiered system you completed in Phase 1, unless you have not completed the 3-tier system, or your 3-tier system is not fully functional).

The project submitted should be in a compressed file (e.g., in .zip format) that contains all your documentations (e.g., written report, source & executable codes/files and any other supporting documents if any). The written report must be in Word or PDF format. Please rename the .zip file to the following format:

<Team-leader's Student ID>_< team-leader's last & first name>_< other team member's ID_ last name>_CSI3344_A3.zip.

If the assignment is done by an individual, please rename the .zip file to the following format:

<Student ID>_<last name>_< first name>_CSI3344_A3.zip.

> *As an example, if your team leader's ID is 12345678, last name is SMITH, and fist name is Ben, and the other team member's ID is 15555555 and his Last name is MAX, then the submission file should be named 12345678_SMITH_ Ben_15555555_MAX_CSI3344_A3.zip. If the assignment is completed by Ben SMITH alone, the submission file should be named 12345678_ SMITH_Ben_CSI3344_A3.zip*

(Note: If your group completed tasks for "Bonus Marks", please pack all the codes into a separate folder, rename that folder "bonus works", and include the folder in the .zip file)

Note that files found to contain viruses or other infecting mechanisms shall be awarded a *ZERO* mark.

Your attention is drawn to the university rules governing cheating/plagiarism and referencing. Please refer to the section of *Academic Integrity and Misconduct* in the next page.

ECU rules/policies will apply to late submission of this assignment.

It is also recommended that you also include a ReadMe.txt text file as a support document, which includes step-by-step procedure for setting up and running the system, as well as user authentication details (if this information is not provided in the user Manual section of the written report).

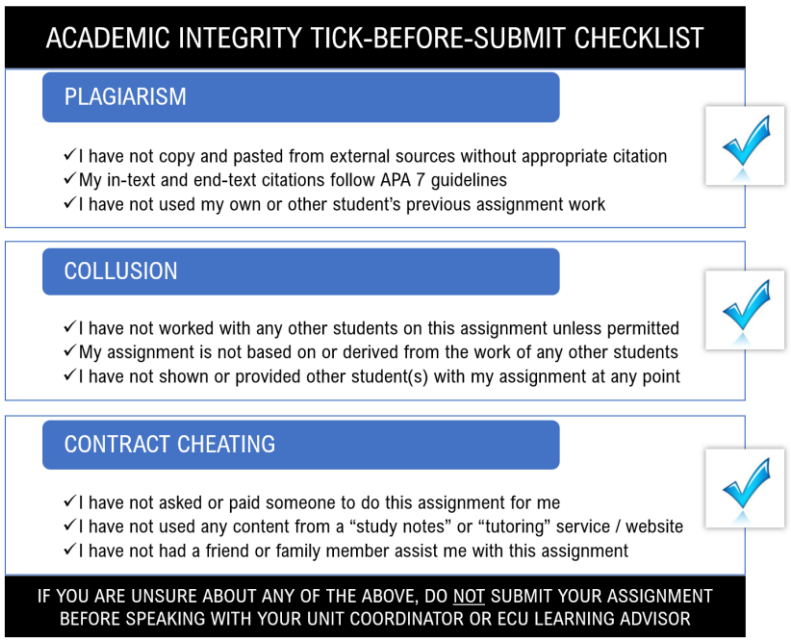
Video submission:
The submission should be one video link (or two video links if you separate the video in two video clips).

Peer review submission (optional):
Complete the form and submit (note that a form template is available on Canvas).

Note that separate submission links are available for each submission.

Academic Integrity and Misconduct

- This assignment is a **group assignment** (for up to two students per team). Your entire assignment must be your own work of the team (unless quoted otherwise) and produced for the current instance of the unit. Any use of uncited content that was not created by your team constitutes *plagiarism* and is regarded as Academic Misconduct. All assignments will be submitted to plagiarism checking software, which compares your submission version with previous copies of the assignment, and the work submitted by all other students in the unit (in the current semester or previous years/semesters).
- *Never give any part of your assignment to someone else or any other group,* even after the due date or the results are released. Do not work on your team assignment with other students or teams. – You can help someone (in other team) by explaining concepts, demonstrating skills, or directing them to the relevant resources. But doing any part of the assignment for them or with them or showing them your work is inappropriate. An unacceptable level of cooperation between students/groups on an assignment is *collusion* and is deemed an act of Academic Misconduct. If you are not sure about plagiarism, collusion or referencing, simply contact your lecturer or unit coordinator.
- You may be asked to explain and demonstrate your understanding of the work you have submitted. Your submission should accurately reflect your understanding and ability to apply the unit content and the skills learnt from this unit.
- Before submitting your assignment 3 (and video demos), team leader should make sure you have checked all items in the **Academic Integrity tick-before-submit checklist** below and watch the video by clicking the link next to the checklist image:



Watch this video before submitting your assignment

Indicative Marking Guide:

	Criteria	Out of marks
Implementation of the project	Phase 1: Two-tier version completed + user authentication, etc.	35
	Phase 2: three-tier version completed and tested.	20
(Project) Bonus mark	(up to 5 marks)	
The report	Executive summary: abstraction of the report & vital information as a whole; Thought/idea organization: conjunctions & cohesion; Clarity: discussion flow and integrity etc.; Citations to *References*; User manual; Test cases; Conclusions achieved; References; Quality of the report: Report presented as per Format requirement; Report length - not too long and too short (e.g., 3000~4000 words, in 8-16 pages?).	14
Submission	Files submitted as per submission requirement.	1
Video demo &reflection	As per requirement	30
Penalty	(possible Plagiarism or Late submission penalty)	
Total	Total mark of 100 (which is then converted to 50% of unit mark)	100 / (50%)

Appendix A: *Example Python-like pseudo-code (for reference only):*

```
"""
File Average.py.
An exemplar Python code.
"""
import random

def  Average (lyst):
        #lyst: an array of unit scores, ended with -1, which is not a valid score
        #double average: average of all scores
        #Initialization phase
        total = 0        #total - sum of scores
        gradeCounter = 0 #gradeCounter: to track the number of scores in the array lyst[])
        #Processing phase
        n=len(lyst)
        i=0
        if (n ==0):
                return 0
        while lyst[i] != -1:
                gradeValue = lyst[i]
                total = total + gradeValue
                gradeCounter +=  1
                i +=1
        if (gradeCounter != 0 ):
                average = total / gradeCounter
                return average
        else:
                return 0

def main(average = Average):
    lyst = [ 10, 20, 30, 40, 50, 60 ,70, 80, 90, 100, -1]
    size = 10
    #for count in range(size):
     #   lyst.append(random.randint(1, 100 ))
    print(lyst)
    a = average (lyst)
    print("The average value is:", a)
    lyst1=[ ]
    while True:
        v = input("Please enter an integer as a unit mark (enter -1 to stop):")
        x=int (v)
        if x !=-1:
            lyst1.append(x)
        else:
            lyst1.append(-1)
            break
    print(lyst1)
    a = average (lyst1)
    print("The average value is: a = ", a)


if __name__ == "__main__":
    main()
```

Appendix B: *Extra work for groups with three students:*

If your group has three team members, your group work must implement server-2 with a real database server (e.g., MySQL server, SQL server, Oracle server, etc.). And your group should also do the following two extra tasks:

- Develop an *ADDITIONAL* mini database interface for server-2 that can be used to helps database administrators (DBAs) to prepare and/or check data stored in server-2.
  - Develop a mini user interface for the database server (note: you can use SQL or other development tool/s) such that
    - it can display the database scheme of the *OSCLR* database (i.e., tables and their structures, including columns information, primary/foreign keys, etc.).
    - it can display all available student learning records stored in the *OSCLR* database.
    - it allows users to enter/delete/edit data records stored in the *OSCLR* database. (*Note*: you should use an integer to keep track of number of data records created so far.
      Be careful – Don't allow more than 30 unit scores/marks for any student.
  - Attach a simple *User Manual* showing how to install your mini *database interface*.

- Develop a "user registration" client that allows a user (for example a non-OUST student) to register as a user of the system (for simplicity, you may only create users that allows <PersonID, password> to authenticate users. A "Non-OUST_users" table may be created and stored in the *OSCLR* database (note that the table/s can only be created by way of server-1 application because client applications have no direct access to the server 2).

In addition to this, you should also add a short video clip (or as a part of the demo video) to demonstrate how you can use the interface to interact with the OSCLR database.

(The End of the Assignment description)