Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code:	CMT115
Module Title:	Programming Coursework Using Python
Lecturer:	Federico Liberatore
Assessment Title:	Written Assessment
Date Set:	22nd October 2019
Submission date and Time:	9th December 2019 at 9:30AM
Return Date:	13th January 2020

This coursework is worth 60% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1. If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;
2. If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Your submission must include the official Coursework Submission Cover sheet, which can be found here:
`https://docs.cs.cf.ac.uk/downloads/coursework/Coversheet.pdf`

Submission Instructions

Description	**Type**	**Name**
Cover sheet	One pdf file	`[Student number].pdf`
Code for Q1	One Python file	`Q1_[Student number].py`
Code for Q2	One Python file	`Q2_[Student number].py`

Any code submitted will be run on a system equivalent to that available on the laptops provided to the students, using Python3, and must be submitted as stipulated in the instructions above. The code should run without any change.

Any deviation from the submission instructions above (including the number and types of files submitted) may result in a mark of zero for the assessment or question part.

Assignment

This coursework asks you to simulate a simple processing system and generate the data for the simulation. The processing system is comprised of three identical processors that process randomly generated tasks.

In the following, a description of the simulation data and of the system is given.

Simulation Data

The data required to run the simulation represents different tasks that need to be carried out by a processor. Each task is characterised by the following properties.

ID : A string of six characters. Each character is randomly chosen (uniform probability) from letters ('a'-'z' and 'A'-'Z'), digits ('0'-'9') and some special characters ('@', '_', '#', '*', '-', and '&').

Arrival : A random real value generated by a uniform distribution from 0 to 100.

Duration : A random value generated by an exponential distribution of parameter 1, rounded up.

Simulated System

The system is comprised of a clock and three identical processors.

At the beginning, the clock is set to zero and the processors are not busy and are, therefore, available. The following message is displayed in the console:

*** SYSTEM INITIALISED ***

As the clock advances, the tasks enter the system at their specified arrival time. In the rare eventuality of multiple tasks arriving at the same time, the processing order is indifferent and the tasks are processed one at the time. Without loss of generality, from now on it is assumed that only one task enters the system.

When a task enters the system, the following message is displayed:

*** [CLOCK] : Task [TASK ID] with duration [TASK DURATION] enters the system.*

where [CLOCK] is the current clock's value, [TASK ID] is the incumbent task's ID and [TASK DURATION] is its duration.

Next, the system checks the ID of the entering task. If the ID does not satisfy at least 3 of the following rules, the task is automatically discarded:

- At least one lowercase letter.
- At least one uppercase letter.
- At least one digit.
- At least one among the special characters.

If a task is discarded, the following message is displayed:

*** Task [TASK ID] unfeasible and discarded.*

Otherwise, if the task's ID passes the filter, the following message is displayed:

*** Task [TASK ID] accepted.*

Then, the task needs to be assigned to a processor. If a processor is available, then the task is assigned to it, the processor is busy for the whole duration of the task and it becomes available when it ends. Otherwise, the task must be put on hold and assigned to the first available processor according to a FIFO strategy.

When a task is put on hold the message displayed is:

*** Task [TASK ID] on hold.*

On the other hand, when a task is assigned to a processor the following message is displayed:

*** [CLOCK] : Task [TASK ID] assigned to processor [PROCESSOR #].*

where [PROCESSOR #] is the processor number, i.e., either 1, 2 or 3. When a task is completed, the message displayed is:

*** [CLOCK] : Task [TASK ID] completed.*

Finally, when all the tasks have been processed and completed, the simulation ends and the following message is displayed:

*** [CLOCK] : SIMULATION COMPLETED. ***

Questions

Provide code to answer the questions below.

1. Write the code that randomly generates a simulation dataset, according to the description provided above. A simulation dataset is comprised of 100 tasks. The code must store the dataset in an SQL database (using sqlite3). Provide the code that creates the SQL database in Python. Please note that the submission does not include any database.

2. Write the code that simulates the processing system according to the description provided above. The simulator should acquire the simulation dataset (tasks) from an SQL database (built using sqlite3) and store it in a queue. The system check on the task IDs must be carried out using regular expressions. At each step of the simulation, the simulation clock should be updated to the next significant event, e.g., task arrival, task processing completion.

Learning Outcomes Assessed

- Use the Python programming language to complete programming tasks.
- Demonstrate familiarity with basic programming concepts and data structures
- Implement Abstract Data Structures in an Object-Oriented style
- Apply regular expressions.
- Design, implement and utilise relational databases.

Criteria for assessment

Credit will be awarded against the criteria of functionality and code quality:

Functionality: Does the code perform the required task correctly and accurately?

Code quality: Is the code elegant and well-written; easy to run; simplified by the use of built-in languages features where appropriate; readable and easy to follow; properly commented? Are appropriate functions defined and written to enable reuse?

The mark breakdown for each part is given in the following.

Question 1

- Functionality: Correct generation of tasks' properties. 10 marks
- Functionality: Correct creation of SQL database and storage of tasks. 5 marks
- Code quality. 5 marks

Question 2

- Functionality: Correct loading of dataset from SQL database. 5 marks
- Functionality: Correct system check on task IDs. 5 marks
- Functionality: Correct use of data structures. 5 marks
- Functionality: Correct execution of the simulation and console outputs. 15 marks
- Functionality: Parsimonious clock update. 5 marks
- Code quality. 5 marks

For each item evaluated, partial marks can be assigned as illustrated in the following table.

	Distinction (70-100%)	Merit (60-69%)	Pass (50-59%)	Fail (0-49%)
Functionality	Fully working functionality achieved using a relevant python approach.	Functionality working with some minor errors.	Functionality presents at least a major error and/or wrong output.	Faulty functionality with wrong implementation and wrong output.
Code Quality	Code is elegant, efficient, with no redundancies, well commented, easy to run, perfectly modular (i.e., appropriate functions and/or classes defined), and makes use of built-in language features.	The code complies with less than six of the characteristics required for distinction.	The code complies with less than four of the characteristics required for distinction.	The code complies with less than two of the characteristics required for distinction.

Feedback

Feedback on your coursework will address the above criteria.

Individual feedback will be returned by the end of week 12.

Group feedback will be provided in the revision lecture in week 12 and via model solutions directly after all student submissions have been made.