Assessment (non-exam) Brief

UCL
SCHOOL OF MANAGEMENT

Module code/name	INST0004 Programming 2
Module leader name	Dr Daniel Onah
Academic year	2023/24
Term	1
Assessment title	Tutorial Sheets 1, 2,3,4,6,8
Individual/group assessment	Individual

Submission deadlines: Students should submit all work by the published deadline date and time. Students experiencing sudden or unexpected events beyond your control which impact your ability to complete assessed work by the set deadlines may request mitigation via the extenuating circumstances procedure. Students with disabilities or ongoing, long-term conditions should explore a Summary of Reasonable Adjustments.

Return and status of marked assessments: Students should expect to receive feedback within one calendar month of the submission deadline, as per UCL guidelines. The module team will update you if there are delays through unforeseen circumstances (e.g. ill health). All results when first published are provisional until confirmed by the Examination Board.

Academic Misconduct: Academic Misconduct is defined as any action or attempted action that may result in a student obtaining an unfair academic advantage. **Academic misconduct includes plagiarism, obtaining help from/sharing work with others be they individuals and/or organisations or any other form of cheating.** Refer to Academic Manual Chapter 6, Section 9: Student Academic Misconduct Procedure - 9.2 Definitions.

Referencing: You must reference and provide full citation for ALL sources used, including articles, textbooks, lecture slides and module materials. This includes any direct quotes and paraphrased text. If in doubt, reference it. If you need further guidance on referencing please see UCL's referencing tutorial for students. Failure to cite references correctly may result in your work being referred to the Academic Misconduct Panel.

Use of Artificial Intelligence (AI) Tools in your Assessment: Your module leader will explain to you if and how AI tools can be used to support your assessment. In some assessments, the use of generative AI is **not permitted** at all. In others, AI may be used in an **assistive** role which means students are permitted to use AI tools to support the development of specific skills required for the assessment as specified by the module leader. In others, the use of AI tools may be an **integral** component of the assessment; in these cases the assessment will provide an opportunity to demonstrate effective and responsible use of AI. See page 3 of this brief to check which category use of AI falls into for this assessment. Students should refer to the UCL guidance on acknowledging use of AI and referencing AI. Failure to correctly reference use of AI in assessments may result in students being reported via the Academic Misconduct procedure. Refer to the section of the UCL Assessment success guide on Engaging with AI in your education and assessment.

:

Content of this assessment brief

Section	Content
A	Core information
B	Coursework brief and requirements
C	Module learning outcomes covered in this assessment
D	Groupwork instructions (if applicable)
E	How your work is assessed
F	Additional information

Section A Core information

Submission date	On the module page
Submission time	On the module page
Assessment is marked out of:	100
% weighting of this assessment within total module mark	100%
Maximum word count/page length/duration	NA
Footnotes, appendices, tables, figures, diagrams, charts included in/excluded from word count/page length?	NA
Bibliographies, reference lists included in/excluded from word count/page length?	NA
Penalty for exceeding word count/page length	No specified maximum (and thus no penalty for exceeding)
Penalty for late submission	Standard UCL penalties apply. Students should refer to Refer to https://www.ucl.ac.uk/academic-manual/chapters/chapter-4assessment-framework-taught-programmes/section-3-moduleassessment#3.12
Submitting your assessment	Weekly at the scheduled deadline/time

Anonymity of identity. Normally, <u>all</u> submissions are anonymous unless the nature of the submission is such that anonymity is not appropriate, illustratively as in presentations or where minutes of group meetings are required as part of a group work submission	The nature of this assessment is such that anonymity is not required.

Section B Assessment Brief and Requirements

On the Tutorial sheets.

C: Section Module Learning Outcomes covered in this Assessment

This assessment contributes towards the achievement of the following stated module Learning Outcomes as highlighted below:

Weekly lectures for INST0004 Programming 2

Section D

: Groupwork Instructions (where relevant/appropriate)

NA

Section E

: How your work is assessed

Within each section of this assessment you may be assessed on the following aspects, as applicable and appropriate to this assessment, and should thus consider these aspects when fulfilling the requirements of each section:

- The accuracy of any calculations required.
- The strengths and quality of your overall analysis and evaluation;
- Appropriate use of relevant theoretical models, concepts and frameworks;
- The rationale and evidence that you provide in support of your arguments;
- The credibility and viability of the evidenced conclusions/recommendations/plans of action you put forward;
- Structure and coherence of your considerations and reports;
- Appropriate and relevant use of, as and where relevant and appropriate, real world examples, academic materials and referenced sources. Any references should use either the Harvard OR Vancouver referencing system (see References, Citations and Avoiding Plagiarism)
- Academic judgement regarding the blend of scope, thrust and communication of ideas, contentions, evidence, knowledge, arguments, conclusions.
- Each assessment requirement(s) has allocated marks/weightings.

Student submissions are reviewed/scrutinised by an internal assessor and are available to an External Examiner for further review/scrutiny before consideration by the relevant Examination Board.
It is not uncommon for some students to feel that their submissions deserve higher marks (irrespective of whether they actually deserve higher marks). To help you assess the relative strengths and weaknesses of your submission please refer to UCL Assessment Criteria Guidelines, located at https://www.ucl.ac.uk/teaching-learning/sites/teaching-learning/files/migratedfiles/UCL_Assessment_Criteria_Guide.pdf

The above is an important link as it specifies the criteria for attaining 85% +, 70% to 84%, 60% to 69%, 50% to 59%, 40% to 49%, below 40%.

You are strongly advised to **not** compare your mark with marks of other submissions from your student colleagues. Each submission has its own range of characteristics which differ from others in terms of breadth, scope, depth, insights, and subtleties and nuances. On the surface one submission may appear to be similar to another but invariably, digging beneath the surface reveals a range of differing characteristics.

Students who wish to request a review of a decision made by the Board of Examiners should refer to the UCL Academic Appeals Procedure, taking note of the acceptable grounds for such appeals.

Section F

Note that the purpose of this procedure is not to dispute academic judgement – it is to ensure correct application of UCL's regulations and procedures. The appeals process is evidence-based and circumstances must be supported by independent evidence.

Section G

: Additional information from module leader (as appropriate)

Work in accordance with the weekly assessment brief and the tutorial sheet questions herein.

Section H

INST0004: Programming 2
Tutorial Sheet 10
Week 10

Academic Year: 2023/24

Set by Module Leader: Daniel Onah

The aim of the practical sheet is to provide you with a set of exercises to practice developing your programming skills. Writing programmes will help you develop programming skills and prepare for the Programming Test and the Moodle Quiz.

Assessed Tutorial Questions:

Some questions in the labs are marked with a [*] symbol. These questions are compulsory and you will be assessed on one or more of these in the following week's lab. For most of you, this means you will be assessed on:

- Tutorial sheet 1 in Lab 2
- Tutorial sheet 2 in Lab 3
- Tutorial sheet 3 in Lab 4
- Tutorial sheet 4 in Lab 5
- Tutorial sheet 6 in Lab 7, and
- Tutorial sheet 8 in Lab 9

Please look at all the questions spend some time thinking carefully about them, before asking for help. If you are still stuck:

- attend the live practical & booster sessions and ask me, or the TAs, for help
- or post a question on the Moodle discussion board.

Once you have completed the assessed questions you should upload the specified files to the Assessment tab on Moodle under the appropriate submission link. You can submit at the specified submission time on Moodle.

Published September 3rd 2022

Section I

In preparation for your 1-to-1 viva, you should be able to clearly indicate which constructs such as classes, functions, variables used in your program is of your own work and which classes or features where from other sources (e.g., from textbooks, online sources etc.).

Please refer to Read about scheduling and live sessions and Read about tutorial based assessment, submissions and vivas for additional information on the procedures of being assessed.

A reminder on how to compile a programme

This is just a quick reminder of how to compile a simple Python program.

1) Make sure you organise your files into folders for each of the weeks:
   a. You should keep your files on the `N:` drive. I suggest creating a folder called `INST0002` with subfolders `week01, week02` etc. for each of the tutorial files.
2) You should open the Command Prompt Shell and navigate to the current directory where your Python files are located, for example, `N:/INST0002/week01`. To do this, use the cd command. For example:

   ```
   cd  N:/INST0002/week01
   ```

   To change the drive from C: to N: simply type in `N:` in the command prompt shell.

   Speak to a TA or a peer student if you need help with using the N drive or the Command Prompt shell.

3) Once in the working directory, you can execute/run the files without providing the path to the file:

   ```
   python hello_world.py
   ```

   (where python is the name of the interpreter that will trigger the Runtime Environment and execute specific file)

Section J

Exercise 1 [] Sales Record

Learning Objectives: [A] input() function [B] method [C] reading and writing to file [D] control statements [E] readlines() [F] Loop constructs [G] list index.

Suppose you are given a sales record (**SalesItemRecord.csv**) and you are required to compute the total sale and average sale into another file (FinalSale.txt).

Problem task:

You should create a class program with its instance for this task. The new file to be written to should contain the sales content from the original file and include the computed total and average sale (study Lecture 10 for a guide).

You are required to create a header called **Total** that would be an additional column to store the total of each item. At the end of the program, the total sale and average of the sales should be displayed at the end of the file.

The SalesItemRecord.csv provided in the Exercise2 folder contains the follow sales record as seen below:

SalesItemRecord

Product	Unit Price	Quantity
Table Lamp	340	20
Television	1900	9
Hair Dryer	158	56
Refrigerator	1970	13
Microwave	200	67
Washing Machine	892	15
Pressing Iron	314	19

You should compute the sales of the individual items and place this in the **Total** column created as a new header in the **TotalSales.csv** file provided.

Description

Section K

Create a class called **SalesRecord** with a constructure that initialise the input and output files. The files should be passed into the instance of the class during the object creation. This means you have to create the class object that would take the two files as arguments.

computeSale()

Create a method called **computeSale()** that would take no parameter list except the self parameter. The method should be used to compute the program file processing. The try and except block should handle any exception that would occur from within the file processing phase.

In this method, you should be able to open and read the input file ready for processing (watch lecture 10 a guide). Since the input file contain headers, you would want to avoid processing them, as this would hinder the processing of strings and numeric values together (this mostly would lead to error). Open and write to the output file **TotalSale.csv**. Ensure you use appropriate write method for csv file format.

Iterate over the rows of the file and compute the total sale of each individual items, compute the total and average sales (see the output session for an insight on how to present this).

One way you could access the content of the input file for the computation is by using the file processing methods such as strip() and split() together with the appropriate index.

In order to add the new contents such as total and average to the file, you can you the append code for file processing (practice/study the week 10 quizzes and watch lecture 10 for guide).

Try/Except

Use try and except block to handle the exception error if the items in the **SalesItemRecord.csv** file is not processed properly. For example when improper usage of index or not defining the index appropriately.

Hint: Ensure you import the cv module and all packages needed for reading the csv file (watch lecture 10 for a guide)

Section L

The output of the programme should be:

```
danny$ python3 sales_record_ex02.py
```

TotalSale

Product	Unit Price	Quantity	Total
Table Lamp	340.0	20	6800.0
Television	1900.0	9	17100.0
Hair Dryer	158.0	56	8848.0
Refrigerator	1970.0	13	25610.0
Microwave	200.0	67	13400.0
Washing Machine	892.0	15	13380.0
Pressing Iron	314.0	19	5966.0
Total Sale			£91104.00
Average Sale			£13014.86

Exercise 2 [] Sales Transaction

Learning Objectives: [A] Function [B] Loop constructs, [C] conditional statements [D]open, read and write methods [E] main() function.

Write a program to demonstrate a point-of-sale transaction. Suppose you have been offered a text file to process and calculate the total sale of the day's transaction, how would you process that file such that all the original content would be process alongside the new output (total sale). In this task, you have been provided with two files **Sales.txt** and **TotalSale.txt.** The sale file contains the following items and their prices.

Radio	230
Book	150

Pen	23
Fan	421
Desk	890

Main() function

You are required to write the program using a main() function. Open and read the input file (**Sales.txt**) and write the context and the total sale after computation to the output file(**TotalSale.txt**). Use a loop construct to iterate over the data and check whether the data is empty and split the data to its unique entity (watch the lecture 10 for a guide). You are required to extract the two data fields in the input file according to their respective data type and write them with the total sale into the output file.

The output of the programme should be:

```
danny$ python3 sale_transaction_ex02.py


Radio                  230.00
Book                   150.00
Pen                     23.00
Fan                    421.00
Desk                   890.00
Total Price:          1714.00
```

Section N

Exercise 3 [] User Define Class Exception

Learning Objectives: [A] print() function [B] conditional statement, [C] raise exception [D] user define exception classes[E] method [F] import.

You are required to write a program that would verify whether user input is greater than number 1. Write a user defined classes to perform exception error handling task. Create three classes for this task, **PaymentList**, **RoomException**, **RuntimeExceptionTester**.

Creating your own exception classes

In this program you will be creating your own exception class. You can create yow own exception class by inheriting from any predefined exception class. Generally speaking, if you wish your own exception class to be unchecked then it should inherit from *RuntimeException* (or one of its subclasses), whereas if you wish your exception to be checked you can inherit from the general **Exception** class. You have been provided with a class description below, we wish to make the exception executed by the constructor ***unchecked***, as it is a logical error not an input/output error. In this case, we will need to define our exception class by inheriting from either a **RuntimeException** class or general **Exception** class. We will call this new exception class **RoomException** so it can be used throughout our program application if need be.

Note:

- As well as inheriting from some exception class, user-defined exception classes should have two constructors defined within them. One should take no parameters and simply call the constructor of the superclass with a default message of your choosing.
- The other constructor should allow a user-defined message to be supplied with the exception object.
- The **PaymentList** program constructor can now be modified to make use of this new **RoomException** class.

Create a tester class called RoomExceptionTester to use the RoomException class and identify any exception error in the program. In the tester class program, remember to add the general except clause to catch any exception that we might not yet have considered. In this except clause you will have to print the stack trace in this error-handler to determine the exact cause of this unexpected error

Section O

Class Development Process

RoomException

This **RoomException** class should inherit a standard built-in class called *Exception*. The class should define a constructor that would take as parameter a message and inclusive of a self parameter. It is also possible to assign the message parameter within the constructor with this message "*Default error message*". Define a super constructor of the parent class or inherited class with the message parameter.

PaymentList

This **PaymentList** class should inherit the method and properties of the **RoomException** class. This class should initialise a constant variable **MAX** to zero. Define a constructor with one parameter **maxIn** including the self parameter. Now declare a conditional statement to check whether the integer input is greater than 1. If the input is less than 1, the program should raise a *RoomException* error with the message "Invalid list of set entry:" and inclusive of the value entered (see the output section for an insight). However, if the input is greater than 1, store the value as the maximum value and display a message. For example "The number 4 is greater than or equal to 1."

RuntimeExceptionTester

Create the last user define class called *RuntimeExceptionTester*. This class should import the **RoomException** class and the ***PaymentList*** class. The class should inherit the **RoomException** to handle the exception error. Create a try and except block to request for user input. The input should be passed as argument into the *PaymentList* constructor. Call the first except block with the user defined *RoomException* class and display the exception message. Create a generic exception handler class using the built-in Exception class and save this as a variable name e.g. '*exception*' etc. This general exception class should handle all other type or syntax errors that would be raised from the program such as when a floating-point number, or strings entered as input value etc. The program should display a message "End of program" after each execution of the program code (see the output and watch the lecture 10 as a guide).

The output of the programme should be:

```
danny$ python3 RuntimeExceptionTester_ex03.py
Enter size of list:7
The number 7 is greater than or equal to 1.
End of program

danny$ python3 RuntimeExceptionTester_ex03.py
Enter size of list:0
Invalid entry for list size: 0
End of program
```

Section P

```
danny$ python3 RuntimeExceptionTester_ex03.py
Enter size of list:-5
Invalid entry for list size: -5
End of program

danny$ python3 RuntimeExceptionTester_ex03.py
Enter size of list: A
invalid literal for int() with base 10: 'A'
End of program
```