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.

Section C: 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.

INST0004: Programming 2
Tutorial Sheet 09
Week 09

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 [] Counting Number of Unique Words Program

Learning Objectives: [A] input() function [B] function [C] return statement [D] control statements [E] Sets [F] Loop constructs.

Problem Statement: Determine the number of unique words that are contained in a specific input text document. This task is the same as the one in Lecture 9.

Procedure 1: Understand the processing task.

To count the number of unique words in a text document, we need to process each word and determine whether the word has been encountered earlier in the document. Only the first occurrence of a word should be counted as being unique.

The easiest way to solve this task is to read each word from the file and add it to a set. Because a set cannot contain duplicates, the add method will prevent a word that was encountered earlier from being added to the set. After processing every word in the document, the size of the set will be the number of unique words contained in the document.

Procedure 2: Decompose the task into modular steps.
This problem can be split into several simple steps. Follow the pseudocode below:

```
Create an empty set.
For each word in the text document.
    Add the word to the set.
number of unique words = size of the set
```

Creating an empty set, adding an element to a set, and determining the size of a set after each word has been added are standard set operations. Reading the words in the file can be handled as a separate task in the program (you can get more guide from Lecture 9 ***counting unique words***).

Procedure 3: Build the set of unique words.

To build the set of unique words, we must read individual words from the file. For simplicity, we use a literal file name. For a more useful program, however, the file name should be obtained from the user as input.

```
inputFile = open("nurseryrhyme.txt", "r")
for line in inputFile :
   theWords = line.split()
   for word in theWords :
      Process word.
```

Important Information
Here processing a word involves adding it to a set of words. In counting unique words, however, a word cannot contain any characters that are not letters. In addition, the capitalized version of a word must be counted as being the same as the

non-capitalized version. To aid in handling these special cases, let's design a separate function that can be used to "clean" the word before it's added to the set.

Procedure 4: Clean the words.

To strip out all characters that are not letters, we can iterate through the string, one character at a time, and build a new clean word using the lowercase version of the character:

```
def clean(string) :
    result = ""
   for char in string :
       if char.isalpha() :
          result = result + char.lower()
  return result
```

Procedure 5: Combine the functions into a single program.

Implement the main function and combine it with the other function definitions in a single file.

Hint: You can create this program using the ***main()*** function or using class and object.

The output of the programme should be:

```
danny$ python3 count_unique_words_ex01.py
Enter filename (enter default as: article.txt): article.txt
The document contains 19 unique words.
```

Exercise 2 [] Insertion Sort Algorithm

Learning Objectives: [A] print() function [B] Loop constructs, [C] return statements [D] list [E] Classes and Objects

Write a program to sort a list of names in a class in descending order. Your program should request for user input and create the class instance object (using the knowledge from lectures 5 and 9).

Insertion Sort class & method

Define a class **InsertionSort** with a class method or a method called **insertion().** This method should have a single parameter list **name** and including a **self** parameter. Use loops to iterate over the ***name*** argument, taking into consideration the length of the **name**. In order to perform the swap, you would need to consider the which assignment operator necessary to perform this. One approach to use, is to assign the iterative values of the loop (e.g., **i** values) to a new variable **j** (see lecture 5 and 9 for a guide). Use a while loop construct to create the loop condition as follows: ***while (j > 0 and name{j – 1] < name[j]).*** This would allow you to create the swap and return the names in descending order. One way to do this, is to assign the name in the **j** position to a **temp** variable. Then assign the next name in the **j – 1** index position to the name in **j** position. Now, assign the **temp** variable to the name in the **j – 1** index position and decrement the value of the **j** iteration.

Object creation

First create a list of names as follows:

name = ["Danny", "Elaine", "Adam", "Jane", "Peter", "Zack", "James", "Oliver"]

Now, create a class instance **insertionObject** with no argument. The next step is to use the class instance to invoke the ***insertion()*** method passing the name as argument.

Algorithm steps:

- Put all the elements of the list to be sorted in the unsorted sub-list.
- Select the first element in the unsorted sub-list and move it to the end of the sorted sub-list.
- Repeatedly swap the last element of the sorted sub-list with the element to its left, until the number to the left is smaller or the element is at the start of the sorted sub-list.
- If the unsorted sub-list still has elements in it go to step 2, otherwise stop.

The output of the programme should be:

```
danny$ python3 implementing_insertion_sort_ex02.py


['Zack', 'Peter', 'Oliver', 'Jane', 'James', 'Elaine', 'Danny', 'Adam']
```

Section M

Exercise 3 [] Selection Sort Algorithm

Learning Objectives: [A] print() function [B] while loop, [C] return statement [D] class and object [E] function [F] Sets

Write a selection sort algorithm program similar to exercise 2 above, but in a reverse manner that would use a different construct to display a set of names in ascending order.

<u>SelectionSort class</u>

Create a class **SelectionSort** that would define a function **selectionSort** with one main parameter list **name** and including the **self** parameter. Within the body of the method, create an empty list **nameSet** declare a while loop construct in confirmation of the set of names. In the loop construct, assign the minimum name to a variable **firstName** (you can use the **min()** function to do this). Add the first name **value** to the empty set **nameSet**.

Create a set of names as follows:

name = **{"Danny", "Elaine", "Adam", "Jane", "Peter", "Zack", "James", "Oliver"}**

<u>Class instance</u>

Create a class instance **selectionObj** and use it to invoke the **selectionSort** method and pass-in the set of names as argument.

Finally, display the names in ascending order. You can use the ***sorted()*** function of Sets to do this.

The output of the programme should be:

```
danny$ python3 implementing_selection_sort_ex03.py


['Adam', 'Danny', 'Elaine', 'James', 'Jane', 'Oliver', 'Peter', 'Zack']
```