Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code	:	CMT309
Module Title	:	Computational Data Science
Lecturer	:	Dr. Oktay Karakus, Dr Luis Espinosa-Anke
Assessment Title	:	Programming Exercises
Assessment Number	:	1
Date set	:	06-11-2023 at 09.00am
Submission date and time	:	14-12-2023 at 11.59pm
Return date	:	25-01-2024

If you have been granted an extension for Extenuating Circumstances, then the submission deadline and return date will be later than that stated above. You will be advised of your revised submission deadline when/if your extension is approved.

If you defer an Autumn or Spring semester assessment, you may fail a module and have to resit the failed or deferred components.

If you have been granted a deferral for Extenuating Circumstances, then you will be assessed in the next scheduled assessment period in which assessment for this module is carried out.

If you have deferred an Autumn or Spring assessment and are eligible to undertake summer resits, you will complete the deferred assessment in the summer resit period.

If you are required to repeat the year or have deferred an assessment in the resit period, you will complete the assessment in the next academic year.

As a general rule, students can only resit 60 failed credits in the summer assessment period (see section 3.4 of the academic regulations). Those with more than 60 failed credits (and no more than 100 credits for undergraduate programmes and 105 credits for postgraduate programmes) will be required to repeat the year. There are some exceptions to this rule and they are applied on a case-by-case basis at the exam board.

If you are an MSc student, please note that deferring assessments may impact the start date of your dissertation. This is because you must pass all taught modules before you can begin your dissertation. If you are an overseas student, any delay may have consequences for your visa, especially if it is your intention to apply for a post-study work visa after the completion of your programme.

NOTE: The summer resit period is short and support from staff will be minimal. Therefore, if the number of assessments is high, this can be an intense period of work.

This assignment is worth 30% 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.

Extensions to the coursework submission date can **only** be requested using the Extenuating Circumstances procedure. Only students with approved extenuating circumstances may use the extenuating circumstances submission deadline. Any coursework submitted after the initial submission deadline without **approved** extenuating circumstances will be treated as late.

More information on the extenuating circumstances procedure and academic regulations can be found on the Student Intranet:

https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/extenuating-circumstances

https://intranet.cardiff.ac.uk/students/study/your-rights-and-responsibilities/academic-regulations

By submitting this assignment you are accepting the terms of the following declaration:

I hereby declare that my submission (or my contribution to it in the case of group submissions) is all my own work, that it has not previously been submitted for assessment and that I have not knowingly allowed it to be copied by another student. I declare that I have not made unauthorised use of AI chatbots or tools to complete this work, except where permitted. I understand that deceiving or attempting to deceive examiners by passing off the work of another writer, as one's own is plagiarism. I also understand that plagiarising another's work or knowingly allowing another student to plagiarise from my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings[1]

Assessment

(1) You have to upload the files mentioned in the Submission Instructions section below.
(2) Failing to follow submitted file names, and file types (e.g. naming your file q1.py instead of Q1.py) will have a penalty of 10 points from your total mark.
(3) The coursework includes different datasets, which are automatically downloaded. Since these files are already with the markers, students do not need to submit these files back.
(4) Changing the txt file names, and developing your codes with those changed file names would cause errors during the marking since the markers will use a Python marking code developed with the original file names.
(5) You can use any Python expression or package that was used in the lectures and practical sessions. Additional packages are not allowed unless instructed in the question. Failing to follow this rule might cause you to lose all marks for that specific part of the question(s).
(6) You are free to use any Python environment, or version to develop your codes. However, you should fill and test your notebook in Google Colab since the testing and marking process will be done via Google Colab.
(7) If any submitted code for any sub-question fails to run in Google Colab, that part of the code will be marked as 0 without testing the code in Jupyter, or any other environment.
(8) It is not allowed to use the input() function to ask the user to enter values.
(9) If a function is asked to be developed, the name and input arguments of that function should be the same as instructed in the paper.

Learning Outcomes Assessed

- Use the Python programming language to complete a range of programming tasks
- Demonstrate familiarity with programming concepts and data structures
- Use code to extract, store and analyse textual and numeric data

Criteria for assessment

Credit will be awarded against the following criteria. Functions are judged by their functionality and additionally, their quality will be assessed.

[1] https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/academic-integrity/cheating-and-academic-misconduct

Mark	Functionality (80%)	Quality (20%)
Distinction (70-100%)	Fully working application that demonstrates an excellent understanding of the assignment problem using relevant python approach	Excellent documentation with the usage of __docstring__ and comments
Merit (60-69%)	All required functionality is met, and the application is working probably with some minors' errors	Good documentation with minor missing comments
Pass (50-59%)	Some of the functionality developed with an incorrect output major errors	Fair documentation
Fail (0-50%)	Faulty application with wrong implementation and wrong output	No comments or documentation at all

The functionality (80%) will be tested via several test cases for the submitted solutions. For each question, at least one test case is shared with the students and these cases will be used to mark the functionality of the solutions. Some hidden/non-disclosed and advanced test cases will be used to mark solutions. Collecting correct returns for shared test cases does not mean getting full/high functionality marks since advanced test cases might return errors or mistaken outputs in your solutions.

The quality (20%) of the solutions will be tested via comments usage and docstring definitions (where applicable, e.g. in functions). Incorrect docstring definitions will not be assessed and over-commenting will also be penalised. Your documentation should be concise and clear.

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned within 4 weeks of your submission date via Learning Central. In case you require further details, you are welcome to schedule a one-to-one meeting.

Submission Instructions

Start by downloading **example_notebook.ipynb** from Learning Central, then answer the following questions. You can use any Python expression or package that was used in the lectures and practical sessions. Additional packages are not allowed unless instructed in the question. You can study answering the questions by filling in the appropriate sections in the given example Jupyter Notebook.

Your coursework should be submitted via Learning Central by the above deadline. You have to upload the following file:

Description		Type	Name
Your solution to all questions	Compulsory	One Python (.py) file	student_no.py

if your student number is c1234567, then the submitted file should have a name of **c1234567.py**.

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

You can submit multiple times on Learning Central. ONLY files contained in the last attempt will be marked, so make sure that you upload final files in the last attempt.

Staff reserve the right to invite students to a meeting to discuss the Coursework submissions.

Testing Your Codes

You are given at least one test case for each question with their desired outcomes. These cases will give you the chance to check/test your implementations of the questions. You use the test cases to make sure that:

- Your function does not crash, that is, there are no Python errors when trying to run the function.

- Compare the results of the test cases to your results. Expected/desired results are given at the end of each case.

Please note that returning the same outputs in the test codes does not assure that you will get full marks. We will use additional test cases (not disclosed) to test your functions.

IMPORTANT: You must make sure that your file executes and does not crash before submitting to Learning Central. Any function that crashes or does not execute will receive 0 marks on the respective (sub)question. Note that the test codes are only provided for your convenience.

IMPORTANT: Carefully extract only useful code blocks from your notebook file. Your submitted .py file should only include, (1) import commands at the beginning, and (2) function definitions for all questions. Please do not include any test case running steps, or any lines of code in the global scope.

Support for assessment

Questions about the assessment can be asked on https://stackoverflow.com/c/comsc/ and tagged with **#CMT309**, or during the practice session in Week 6.

Please also use Dr Karakus' and TAs' office hours to discuss concerns on any part of the assessment.

PLEASE SEE THE NEXT PAGE FOR ASSESSMENT QUESTIONS

Q1.) Basic Programming (5 marks)

(This will test your general Python and programming knowledge for creating a function and utilisation of some techniques of strings, lists, etc.)

Write a function `all_rounder(args)` that performs passed sequence method of a given sequence. The inputs will be in form, e.g. `all_rounder([1, 5], 'append', [3, 4, 2, 104])`. This function should create a list variable to store `[1, 5]`, then `.append()` the second list of `[3, 4, 2, 104]` to it. The final list should be returned and will be in a form of `[1, 5, [3, 4, 2, 104]]`.

The input argument storing variable `args` could include any kind of sequence types of `str`, `list`, `tuple` and `dict`. Depending on the passed sequence type, `args` should also include a method, and depending on the method some more entries.

The function `all_rounder()` should only perform one operation at a time. Test cases will never pass more than one sequence type and more than one sequence method.

Your function should return a message when a non-existing sequence method entered, and must not return an error.

Hint 1: Using `eval()`, `compile()`, `exec()` might be useful.

Hint 2: Avoid using `try` … `catch` and some extra libraries. This question solely be created via basic Python commands.

An example test case is given below:

```
>>> all_rounder('#', 'join', '(\'Jack\', \'Mahmut\')')
'Jack#Mahmut'
```

Q2.) NumPy (5 Marks)

(This will test the general `numpy` knowledge with a math related question.)

Write a function `padded_broadcasting(func, a, b, pad)` that takes the arguments `func`, `a`, `b` and `pad`. Here `a` and `b` are the `numpy` arrays and `func` represents the arithmetic function (e.g. `np.add`). `pad` represents the number used for padding.

For instance, if `pad=2` then the smaller array gets padded with 2's.

The function returns a `numpy ndarray` which is the result of the operation `func` applied to the two arrays using appropriate padding. The solution needs to work with all basic mathematical operations, that is addition, subtraction, multiplication, division, and taking a power needs to be implemented.

`a` and `b` can have a different number of dimensions.

`a` and `b` can have mismatches on one, multiple, or even all dimensions, e.g. `(6, 2, 3)` and `(5, 3, 2)`.

You may have to pad `a`, or `b`, or both arrays. Do not apply any padding if it is not necessary i.e. when standard broadcasting works.

The default argument for `pad` should be 1.

An example test case is given below:

```
>>> a = np.asarray([[1,2],[-9, 4]])
>>> b = np.asarray([[3,4,5,6],[1,1,0,-5]])
>>> padded_broadcasting(np.add, a, b, -100)
```

```
array([[   4.,    6.,  -95.,  -94.],
       [  -8.,    5., -100., -105.]])
```

Q3.) NumPy 2 - txt file analysis (5 Marks)

(For this question, a .txt file will be given. You are going to use `numpy` methods to load, read and process the file for calculations, e.g. counting the number of a given word.)

Write a function `txtanalyser(fname, t, f, sel)` that takes a `.txt` file name (`fname`) of football commentary as its input and performs various analyses for the target word (`t`).

The developed `txtanalyser(fname, t, f, sel)` function is expected to load the .txt file via `numpy` functions (`np.loadtxt()`).

The loaded file in fact has two columns for each line seperated by tabs: `'Minute'` and `'Commentary'` where the former refers to the minute in the game and the latter is the commentary in that minute. And example visual of your txt file is

```
90  Joel Matip relieves the pressure with a clearance
90  Emerson Royal puts in a cross...
90  Safe hands from Hugo Lloris as he comes out and claims the ball
```

Following this, the function should carry on funding each repetition of the target word `t` and perform calculation depending on the given (`f, sel`) pair:

- Function `f` can be any `numpy` function and should be in form `np.`, e.g. `f = np.max`.
- if `sel` is `'count'`, then `f` should automatically be assigned to `np.sum` whatever passed for `f` as the function argument. Then, for this selection, your function should calculate the number of times the target word `t` appears in the commentary column of the txt file.
- if `sel` is `'find'`, then your function should calculate the passed `f` function for the minutes column of the txt file. *Hint: Average minutes where 'RED-CARD' word was in the commentary –> `f` is `np.mean` and `t` is `'RED-CARD'`.*

Rules: - Using file I/O functions is **not permitted** - Using `pandas` for data handling and processing **is not permitted**.

An example test case is given below:

```
>>> txtanalyser('commentary.txt', 'RED-CARD!', np.mean, 'find')
59.8
```

Q4) NumPy & regex (10 marks)

Find commentary in alphabetical order (8 Marks)

In this question, your goal is to find all the commentaries whose words are arranged so that their first letters are in alphabetical order. For example:

- `Bernardo Silva walks` (because `b, s, w`)

but not

- `Bernardo Silva leaves` (because `b, s, l`, i.e., `l` comes before `s` in the alphabet)

You must implement a function called `find_alphabetical_order(fpath, check_ties=True)` that takes in two arguments:

- `fpath`: A `string` pointing to the `commentary.txt` file. Note that your function is only expected to handle the format in this file, i.e., `minute<tab>commentary`.
- `check_ties`: A `boolean` value. If set to `True`, your function must handle cases when the first letter of two consecutive words is *the same*. For example, `rearranging recovery`. Specifically, if `check_ties` is `True`, you must continue making pairwise checkings between letters in the word until you find a tie breaker (in the example, you would check first the `r`, which is a tie, then the `e`, which is another tie, then `a` vs `c`, since `a` is before `c`, we can determine that `rearranging recovery` is in alphabetical order. If this flag is set to `False`, your code

Notes:

- Your code must only look at words, not punctuation marks. In a sentence `this is a sentence, do I like it? I think so!`, you must clean out `,`, `?` and `!`.
- Your code must handle special characters like *accents* or the `Ñ` letter (like `árbitro`, `reducción` or `España`, these are Spanish words). This is because some comments may not be in English. The way to handle accents is to look for them and replace them with the non-accented version, or with an `N` in the case of `Ñ`. You must handle both upper and lower case occurrences of these characters.
- You may create additional functions to help break down some of the functionalities outside of `find_alphabetical_order()`.
- **IMPORTANT: You must use regular expressions in at least one** of the following steps:
  - clean the text from punctuation
  - turn Spanish characters into English characters
  - comparing letters

An example test case is given below:

```
>>> res = find_alphabetical_order('commentary.txt', check_ties=True)
>>> print(res[0])

Dimitrios Giannoulis hand-balls.
```

Using values in `res` answer the questions below (2 Marks)

- **Commentary languages (1 Mark)**: How many English comments did we find after calling `find_alphanumerical_order(PATH_TO_COMMENTARY_FILE, check_ties=True)`?
- **Name 3 footballers (1 Mark)**: Name 3 footballers that appear in comments returned by `find_alphanumerical_order(PATH_TO_COMMENTARY_FILE, check_ties=True)`.

Q5) Pandas (5 marks)

(Testing pandas usage and one-liner queries.)

Create a function `pd_query()` that takes no input arguments. This function should *locally* load `superheros.csv` file which stores power-related information about Marvel superheros.

Rules:

- The function `pd_query()` should implement 4 queries below and **return a data frame for each** in a `tuple`.
- All pandas query should be a **ONE-LINER**. Implementations more than 1-line will not be evaluated.

Queries:

For each, return a dataframe which stores

- (1 mark) superheros who are *good*, has greater than 80 intelligence but lower than 20 speed.
- (1 mark) average values of each column for superheros with durability value of 120.
- (1 mark) average Total values of good and bad superheros.
- (2 marks) names of superheros who has at least 3 column values better than Iron Man.

An example test case for calling the function:

```
>>> a,b,c,d = pd_query()
```