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	:	04-11-2022
Submission date and time	:	16-12-2022 at 9:30am
Return date	:	03-02-2023

Extenuating Circumstances submission deadline will be 1 week after the submission date above.

Extenuating Circumstances marks and feedback return will be 1 week after the feedback return date above.

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 can be found on the Intranet: `https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/extenuating-circumstances`

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 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]

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

Assessment

(1) You have to upload the files mentioned in 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 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 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 input() function to ask user to enter values.
(9) If a function is asked to be developed, the name and input arguments of that function should be as 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.

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

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 **Q1.py**, **Q2.py** and **Q3.py** 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.

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

Description		Type	Name
Your solution to question 1	Compulsory	One Python (.py) file	Q1.py
Your solution to question 2	Compulsory	One Python (.py) file	Q2.py
Your solution to question 3	Compulsory	One Python (.py) file	Q3.py

Make sure to include your student number as a comment in all of the Python files! 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 all 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 with three Python codes, named testQ1.py, testQ2.py and testQ3.py. These codes will give you the chance to test your implementations of the questions. Each code runs your implementations for a number of testcases. You use the test codes to make sure that:

- Your function does not crash, that is, there is no Python errors when trying to run the function.
- Compare the results of the testcases to your results. Expected results are given at the end of each code.

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

How to use the test codes:

- Create your functions in any environment.
- In each code, replace `pass` command with your implementation of each required function.
- Execute the updated test file. It will run your implementations for a number of testcases specified in the questions. If any errors occur, you need to correct your code.

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.

Question 1 - Python Programming (25 marks)

1.a. Shift vowels by two positions (13 marks)

Write a function `shift_vowels(s)` that takes as input a string `s` representing a word, sentence, or paragraph. Your task is to shift the vowels by two positions contained in the string, where vowels are the symbols a e i o u (and their capitalised versions A E I O U).

Instructions:

- Shift the vowel by one step using the order: a → e → i → o → u.Some examples are
  If the input is `"a cat"`, the output is `"i cit"`.
  If the input is `"kite"` the output is `"kuto"`.
- When the input is a `"u"`, wrap around. For instance, `"cut"` becomes `"cet"`.
- If a vowel is repeated multiple times the vowel is shifted by multiple steps. The number of steps is equal to the number of repeated vowels + 1. For instance, `"mood"` becomes `"meed"`.
- Preserve capitalisation in the string when producing the output string.
- However, for determining how many steps to shift the vowel, capitalisation is ignored. For instance, the string `"Oops"` contains two repeated `"o"`"s and therefore becomes `"Eeps"` with capitalisation preserved.

1.b - Sum of digits (12 marks)

Write a function `sum_of_digits(s)` that takes as input a string `s` that contains some numbers. The function calculates the sum of all the digits in the string, ignoring any symbols that are not digits.

Example: `sum_of_digits("123")` should return `6` since `1+2+3 = 6`.

Example 2: `sum_of_digits("10a20")` should return `3` because `1+0+2+0 = 3`.

Detailed instructions:

- if `s` includes both digits and nondigits
  - Calculate the sum of digits and return the result whilst ignoring any non-digit symbols in the string.
  - print e.g. for `sum_of_digits("10a20")`
    `'The sum of digits operation performs 1+0+2+0'`
  - Save the extracted non-digits in a variable of interest as a `list` and print
    `"The extracted non-digits are: ['a']"`
- If `s` is not provided or an empty string return `0` and print
  `'Empty string entered!'`.
- If `s` is provided, but it contains no digits return `0` and print
  `'The sum of digits operation could not detect a digit!'`
  `'The returned input letters are: [ALL NONDIGITS HERE]'`
- The returned number should be an integer.

Example 1: When you run `sum_of_digits("a1w3")`, the function should print

```
The sum of digits operation performs 1+3
The extracted non-digits are:  ['a', 'w']
4
```

Example 2: When you run `sum_of_digits("united")`, the function should print

```
The sum of digits operation could not detect a digit!
The returned input letters are:  ['u', 'n', 'i', 't', 'e', 'd']
0
```

Question 2 - Linear algebra via `numpy` (35 marks)

This question has been created to test your `numpy` skills in a well-known linear algebra topic - **system of linear equations**. You are asked to develop three different functions in each sub-sections below.

2.a. Matrix Multiplication (20 marks)

You are asked to write a function `matrix_multiplication(*argv)` that calculates multiplication of **two or more** matrices. Since the purpose of this function is to multiply more than two matrices, the number of arguments is variable. Thus, the function will take `argv` as input of variable number of matrices. Please refer to lecture notes-videos about matrix multiplication operation.

Instructions

- You need to test rules of this operation, and in case of a violation, your function should print `'Matrix dimension mismatch'` and return `None`.
- In cases when there is no violation, your function should print `'Multiplication is successful.'` and return the resulting matrix, `res`.
- This function will not use any ready-to-use matrix multiplication functions either in `numpy` (e.g. `.dot()` or `@`) or in any other modules. You have to develop your own algorithm by using the general programming skills.
- Each input matrix consisting `argv` should be `numpy` arrays. Entries in `list` or ‘ `tuple` will not be accepted.
- The returned matrix `res` should also be a `numpy` array.

2.b. Solving system of linear equations (9 marks)

After having the matrix multiplication function above, you are now asked to write a function `linear_solver(A, b)` to solve system of linear equations. In mathematics, a system of linear equations (or linear system) is a collection of one or more linear equations involving the same variables. An example is given as

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= 3 \\ 0.5x - 4y + 1.5z &= -7 \end{aligned}$$

where we have 3 equations and 3 variables x, y and z. Assuming each parameter of each variable consists of a matrix of variable, we have $A = \begin{bmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ 0.5 & -4 & 1.5 \end{bmatrix}$ and we also have $\chi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, and $b = \begin{bmatrix} 1 \\ 3 \\ -7 \end{bmatrix}$, which gives us the system of linear equation in matrix-vector form as $A\chi = b$. Then, the solution can be found via

$$\chi = A^{-1}b$$

Instructions:

- While function argument `A` is a matrix, `b` is a column vector!
- The function return the solution of variables in a variable `res`, which is a column vector as well.

- All of `A`, `b` and `res` should be `numpy` arrays.
- For the cases below, the function should return `None` and print a suitable message depending on the case below.
  - For number of equations are lower than the variables, the system becomes an **Underdetermined system**, and this system has infinitely many solutions.
  - For number of equations are higher than the variables, the system becomes an **Overdetermined system**, and this system has no solution.
  - Other kinds of dimension mismatches.
- In cases when number of equations and variables are equal, the system has **a unique solution** and can be found via the equation given above. This solution should be returned in variable `res`, and the function should print a suitable message.
- For the matrix inverse, you are free to use `np.linalg.inv()` function.
- Matrix multiplication should be done by using the function, `matrix_multiplication()`, created in the subsection above.

2.c. Pseudo-inverse of a matrix and Least-square solution (6 marks)

Least squares method is another approach to find solution of unknown parameters of a system. The most comment LS estimation is the OLS method that minimizes the sum of squared residuals, and leads to a closed-form expression for the estimated value of the unknown parameter vector χ

$$\chi = A^{+}b$$

where $A^{+} = (A^{T}A)^{-1}A^{T}$ is the Moore–Penrose inverse of matrix A.

You need to write a function `LLS(A, b)` to calculate the unknown parameter vector above.

Instructions:

- The Moore–Penrose inverse, and other matrix multiplications should be done by using the function, `matrix_multiplication()`, created in the subsection above.
- For transpose `.T` and for inverse you are free to use `np.linalg.inv()` functions.

Question 3 - Text analysis via `regex` (40 Marks)

A noun phrase is a word or group of words where the core element is a noun. For example, `'table tennis'` is a noun phrase because it is a sequence of two nouns (`'table'` and `'tennis'`); `'red carpet'` is a noun phrase because it is a sequence of an adjective (`'red'`) and a noun (`'carpet'`), and the noun is the core element of the sequence. However, `'go for a walk'` is not a noun phrase because the core element is a verb (`'go'`).

In this exercise, your task is **find and process all noun phrases in Alice in Wonderland**, by Lewis Carroll. You are provided with two text files, `alice_words.txt` and `alice_tags.txt`.

- `alice_words.txt` contains one *sentence* per line from the book Alice in Wonderland, where each sentence has a sequence of *words* separated by a special separator token `<SEP>`. The first three lines of this file look like this:

```
[<SEP>Alice<SEP>'<SEP>s<SEP>Adventures<SEP>in<SEP>Wonderland<SEP>by<SEP>
Lewis<SEP>Carroll<SEP>1865<SEP>]
CHAPTER<SEP>I<SEP>.
Down<SEP>the<SEP>Rabbit<SEP>-<SEP>Hole
```

- `alice_tags.txt` contains one *tag sequence* per line corresponding to sentences from `alice_words .txt`, where each line has a sequence of *tags* separated by a special separator token `<SEP>`. The first three lines of this file look like this:

```
J<SEP>N<SEP>P<SEP>N<SEP>N<SEP>I<SEP>N<SEP>I<SEP>N<SEP>N<SEP>C<SEP>N
N<SEP>P<SEP>.
I<SEP>D<SEP>N<SEP>:<SEP>N
```

These two files are aligned (i.e., the first line in `alice_words.txt` contains the first sentence of the book, and the first line in `alice_tags.txt` contains the sequence of tags corresponding to the first sentence of the book, and so on). Your task is to use these two files for finding **noun phrases** in `alice_words.txt`. We will consider a noun phrase to be a sequence of one or more nouns (which are denoted with the letter `N`) preceded by zero, one or more adjectives (which are denoted with the letter `J`).

3.1 - Read both files into usable Python objects (12 marks)

You must write Python code that reads both files and stores their contents in memory. You should implement two functions, as follows:

- `gettagssequence(tags_file)` - Reads the contents of `alice_words.txt`, and stores them in a list of lists, where each inner list contains one word per line, lower cased. The separation between words is given by the special token `<SEP>`. So, the first sentence:
  - `[<SEP>Alice<SEP>'<SEP>s<SEP>Adventures<SEP>in<SEP>Wonderland<SEP>by<SEP> Lewis<SEP>Carroll<SEP>1865<SEP>]`

  would be stored as
  - `['[', 'alice', "'", 's', 'adventures', 'in', 'wonderland', 'by', 'lewis', 'carroll ', '1865', ']']`
- `getwordssequence(words_file)` - The contents of `alice_tags.txt` should be stored in a list of strings, where each string is the sequence of lower cased tags (after removing the `<SEP>` special character). So, the first sentence:
  - `J<SEP>N<SEP>P<SEP>N<SEP>N<SEP>I<SEP>N<SEP>I<SEP>N<SEP>N<SEP>C<SEP>N`

  would be stored as
  - `jnpnnininncn`

3.2 - Find all noun phrases (18 marks)

First, write a regular expression that captures the following requirement: a noun phrase is a sequence of one or more nouns (which are denoted with the letter `N`) preceded by zero, one or more adjectives (which are denoted with the letter `J`). (remember to handle changes in capitalization)

Then, use this regular expression to produce a list of lists of tuples, where each inner tuple contains the start and end position (note that end position is **non inclusive**) of each noun phrase your regex found. For example, for the third sentence, the corresponding list of tuples could be `[(2, 3), (4, 5 )]`, because it found two different noun phrases (in the form of tuples):

- `('rabbit',)`

- ('hole',)

Implement this functionality in the `find_positions(tags_sequences)` function, which will take as input the `tags_sequences` variable from the previous step.

3.3 - Print all noun phrases that contain an input word (10 marks)

With the `matches_list` produced in step 2, you can now query your list of sentences contained in `words_sequences` for any noun phrase. In this final part, you implement a function `find_noun_phrase (input_word, matches_list, words_sequences)` that takes as input one input word (a string), and the variables `matches_list` and `words_sequences`, and prints all noun phrases containing that word as well as the sentence number (starting counting from one, and not by zero, which would be the default in Python) in which they appear, formatted like this: `sentence_id:noun_phrase`.

For example, given the word `disappointment`, your function could print:

```
14:great disappointment
```

Note, if you are not able to solve 3.2, you can simply load the provided `matches_list` file, which will allow you to complete this part. To do this, simply run:

```
from ast import literal_eval

matches_list = [literal_eval(line.strip()) for line in open('matches_list.txt')]
```

Support for assessment

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