Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code:	CMT120
Module Title:	Fundamentals of Programming
Lecturers:	Federico Liberatore, Martin Chorley, Natasha Edwards
Assessment Title:	Programming Challenges
Date Set:	1st November 2021
Submission date and Time:	13th December 2021 at 9:30AM
Return Date:	24th January 2022

This assignment is worth 40% 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

All coursework should be submitted via upload to Learning Central.

Description	**Type**	**Name**
Cover sheet	`.pdf` file	`[Student number].pdf`
Python Code	1 `.py` file	`[Student number].py`
JavaScript Code	1 `.js` file	`[Student number].js`

Any code submitted will be run on a system equivalent to the University provided Windows laptop, and must be submitted as stipulated in the instructions above. The code should run without any changes being required to the submitted code, including editing of filenames.

Any deviation from the submission instructions above (including the number and types of files submitted) may result in a deduction of 25% for that question or question part.

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

Assignment

To complete this coursework, you must complete a set of programming challenges in Python and JavaScript.

Each challenge can be awarded a maximum of 10 marks. Therefore, perfectly solving five exercises will give you 50 marks (pass), and perfectly solving all the exercises will give you 100 marks. An exercise is solved perfectly only if high-quality functional code is submitted in both Python and JavaScript. Providing high-quality functional code in only one programming language results in a lower mark (i.e., five marks out of ten). Therefore, you can still pass the coursework by only completing problems in one language, or by completing half the problems in both languages.

You might not be able to solve all the exercises. *This is fine.*

The challenges are described in detail below, and you are also provided with a set of test cases that will check whether your code produces the required output or not. In particular, you will be given two test cases per exercise. You should make sure that your submitted code passes the supplied tests to ensure it functions correctly. However, please note that your code will be tested against further/different test cases. Specifically, each exercise will be tested against four test cases, including the two provided. You should therefore ensure that you try to cover all possible inputs and that your code still functions correctly.

Instructions for completing the challenges

- You will find template code for the assignment on Learning Central. This provides two folders, `python` and `js`. Inside each folder you will find a `template.{js/py}` file, in which you should complete your solutions. You will also find a `test_template.{js/py}` file containing the test cases that will check your code's functionality, along with a folder of test data required for some of the tests. You are also supplied with a `Readme.md` file containing detailed instructions on how to run the test cases to check your code.

- In the templates, the functions' interfaces are given but the functions' bodies are empty. Solve the exercises by correctly filling in the functions' bodies.

- It is forbidden to change the functions' interfaces. However, new functions can be defined to support the solution of the exercises. These functions must have names that are different from those already present in the templates.
- You are NOT allowed to import any additional modules.
- In all the exercises, you can assume that the inputs are provided in the appropriate format and type. Therefore, error-checking is not needed.

You will be given marks for solving each problem in both programming languages. Further marks will be awarded for solution style and quality. The mark scheme is described in further detail later.

Exercise 1: Defying Gravity

Write a function that allows a user to introduce the distance d (in meters) travelled by an object falling, or the time t (in seconds) taken for an object to fall, but not both. The function should return the corresponding falling time t or distance d, respectively. The equations to be used are provided below:

$$d = \frac{1}{2}gt^2$$

$$t = \sqrt{\frac{2d}{g}}$$

where g is the gravitational acceleration and, for the sake of this exercise, it can be considered equal to $g = 9.81$ (m/s^2).

Complete function `freeFall(val,isD)`, taking as input a float, `val`, and a bool, `isD`. When `isD==True`, `val` represents the value of d; otherwise, `val` represents the value of t. The function should return the result of the appropriate equation given above, rounded to the second decimal digit.

Examples:

- `freeFall(1,true)` should return `0.45`.
- `freeFall(0.45,false)` should return `0.99`.

Exercise 2: Rock-Paper-Scissor

Rock-Paper-Scissors (RPS, in short) is a hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes are *rock* (a closed fist), *paper* (a flat hand), and *scissors* (a fist with the index finger and middle finger extended, forming a V). A game has only two possible outcomes: a draw, or a win for one player and a loss for the other. A player who decides to play *rock* will beat another player who has chosen *scissors* ("rock crushes scissors"), but will lose to one who has played *paper* ("paper covers rock"); a play of *paper* will lose to a play of *scissors* ("scissors cuts paper"). If both players choose the same shape, the game is tied and is usually immediately replayed to break the tie (Wikipedia contributors, 2021d).

Write a function that, given a player's strategy for a series of games of RPS, will return the winning strategy.

Function `RPS(s)` takes as argument a string `s`. The string can only contain the letters `R`, `P`, or `S`, representing *rock*, *paper*, and *scissor*, respectively. Each letter corresponds to the shape chosen by the other player on each game. The function should return a string of shapes that wins every single game.

Examples:

- `RPS('RPS')` should return `'PSR'`.
- `RPS('PRSPRR')` should return `'SPRSPP'`.

Exercise 3: List to String

In this exercise you are required to convert an input list into a string. The input list can only contain two types of items: a single letter (i.e., A-Z and a-z, case sensitive) or another list with the same properties. The output string should be constructed in such a way that:

- It starts and ends with square brackets (i.e., `[` and `]`).
- The items of the list that are letters are simply concatenated to the string.
- The content of a sublist should be placed in between square brackets.

Complete function `list2str(l)`, which takes as input a list satisfying the above properties and provides as output a string, as described above.

Examples:

- `list2str([‘a’,[‘b’,‘c’]])` should return `‘[a[bc]]’`.
- `list2str([‘a’,[‘b’,[‘c’]]])` should return `‘[a[b[c]]]’`.

Please, note that the output of the function is a string, rather than a list, as it is quoted.

Exercise 4: Text Preprocessing

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The goal is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them (Wikipedia contributors, 2021c).

Traditionally, before applying advanced NLP techniques, a document must be preprocessed.

Complete function `textPreprocessing(text)` which applies the following preprocessing pipeline to a string `text`:

1. Removal of all punctuation marks `.?!,:;-[]{}()’"`. E.g., the string `‘Hi! The cats are playing’` becomes `‘Hi the cats are playing’`).
2. Conversion of text to lower case (e.g., `‘hi the cats are playing’`).
3. Segmentation into a list of words (e.g., `[‘hi’,‘the’,‘cats’,‘are’,‘playing’]`).
4. Removal of stopwords. Stopwords are words that are so common that do not much information and, therefore, can be removed. For this exercise, make use of the following list of stopwords:

   i, a, about, am, an, are, as, at, be, by, for, from, how, in, is, it, of, on, or, that, the, this, to, was, what, when, where, who, will, with

   (e.g., the result of this step on the sample sentence would be `[‘hi’,‘cats’,‘playing’]`)
5. Stemming, which aims at reducing different forms of a word into a common base form. For the sake of this exercise, you are required to apply a very crude stemmer, based on the following rule: if a word ends in `-ed`, `-ing`, or `-s`, remove the ending (e.g., `[‘hi’,‘cat’,‘play’]`).

The function should return the result as a list of strings.
Examples:

- `textPreprocessing('I think, therefore I am.')` should return `['think','therefore']`.
- `textPreprocessing('When life gives you lemons, make lemonade.')` should return `['life','give','you','lemon','make','lemonade']`.

Exercise 5: Dictionary Dominance

Complete function `isGreaterThan(dict1,dict2)` that takes as input two dictionaries in Python, or two Objects in JavaScript, having only string keys and numerical values. The function returns `True` if and only if `dict1` is greater than or equal to `dict2` with respect to all the keys, and it is strictly greater than `dict2` in at least one key.
Examples:

- `dict1={'a':1,'b':2}` and `dict2={'a':1,'b':1}`. In this case, `dict1` is equal to `dict2` with respect to `a`, but it is greater with respect to `b`; therefore, the function should return `True`.
- `dict1={'a':1,'b':1}` and `dict2={'a':1,'b':1}`. In this case, `dict1` and `dict2` are equivalent; therefore, the function should return `False`.
- `dict1={'a':1,'b':0}` and `dict2={'a':0,'b':1}`. In this case, `dict1` is greater than `dict2` with respect to `a`, but it is lower with respect to `b`; therefore, the function should return `False`.

The two dictionaries/objects might not necessarily have the same keys. If a dictionary/object does not have a key that the other one has, the former is lower than the latter with respect to that key, regardless of the value that the latter might have.
Examples:

- `dict1={'a':1,'b':2,'c':-10}` and `dict2={'a':1,'b':1}`. `dict1` is greater than `dict2` with respect to all the keys; therefore, the function should return `True`.
- `dict1={'a':1,'b':1}` and `dict2={'c':0}`. In this case, `dict1` is not greater than `dict2`, as `dict1` does not have the key `c`; therefore the function should return `False`.

Exercise 6: Reading CSV Files

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas (Wikipedia contributors, 2021b).

Write function `CSVsum(filename)` that reads a CSV file with or without a header, and provides as output a list of the sums of each column.

Example: Suppose that the following is the content of file `test.csv`:

```
var1,var2,var3
1.0, 2.0, 3.0
4.0, 1.0,3.0
0.0,5.0, 0.0
```

which translates into the following table:

var1	var2	var3
1.0	2.0	3.0
4.0	1.0	3.0
0.0	5.0	0.0

Then, `CSVsum('test.csv')` should return `[5.0,8.0,6.0]`.

Exercise 7: String to List

This exercise is basically the opposite of Exercise 3.

You are now required to convert an input string of letters and square brackets (i.e., `[` and `]`) into a list of letters and lists. The square brackets identify where a list starts and ends, while each letter translates into an element of the corresponding list. Read the description of Exercise 3 and see the examples below for more information.

Complete function `str2list(l)`, which takes as input a string satisfying the above properties and provides as output the corresponding list.

Examples:

- `str2list('[abc]')` should return `['a','b','c']`.
- `str2list('[a[bc]]')` should return `['a',['b','c']]`.

Exercise 8: Spacemon Competition

Spacemons are spirits that come from other planets of our star system. When two spacemons meet, they feel the urge to fight until one or them is defeated. Warlocks conjure, tame, and train teams of spacemons, to make them compete in a spectacular tournament.

Note: the paragraph above is a work of fiction.

In this exercise, you are required to complete the function `spacemonSim(roster1,roster2)`, which simulates the result of a competition between two teams of spacemons, `roster1` and `roster2`; the function returns `True` if `roster1` wins, or `False` otherwise.

Disclaimer: no spacemon was harmed in the making of this exercise.

A spacemon is represented as a three-element tuple: `(planet, energy, power)` in Python, in JavaScript as a three-element array: `[planet, energy, power]` where `planet` represents the type of the spacemon, `energy` is its stamina, and `power` is its ability to reduce another spacemon's energy. A roster is simply a list of spacemons.

The `planet` of a spacemon is particularly important as certain types are stronger/weaker against others, as represented in Table 1.

att \ def	Mercury	Venus	Earth	Mars
Mercury	×1	×2	×1	×0.5
Venus	×0.5	×1	×2	×1
Earth	×1	×0.5	×1	×2
Mars	×2	×1	×0.5	×1

Table 1: Attack multipliers depending on type.

In the table, the rows correspond to the attacking spacemon, the columns correspond to the defending spacemon. The cells show the multiplier that must be applied to the attacker's `power` to determine how much energy the defender loses.

A competition is divided into one-on-one matches. Spacemons take part in the competition according to their position in the roster. Therefore, the first match is between the first spacemon of each roster. The spacemons take turns to attack, with the first roster always attacking first. When a spacemon attacks, the total damage inflicted on the opponent is: `damage = type_mult * power`, where `type_mult` is the multiplier specified in Table 1. The `damage` is then subtracted from the opponent's `energy`. If a spacemon's energy drops to 0 (or less), the spacemon is defeated and the match ends. Then, a new match starts between the winner of the previ-

ous match and the next spacemon in the opponent's roster. The winning spacemon does not recover any lost `energy` between matches; also, the first spacemon to attack is, again, the one from the first roster.

Example: Let us consider the following rosters.

- `roster1` is comprised of `('Earth',100,10)` and `('Earth',100,10)`.
- `roster2` is comprised of `('Mercury',80,10)` and `('Venus',80,10)`.

In the first match, the Earth spacemon defeats the Mercury spacemon; however, it loses 70 points of `energy`. In the second match, the former winner is defeated by the Venus spacemon, which receives 10 points of `damage`. Finally, the Venus spacemon wins the third match, losing 35 points of `energy`. The second roster wins the competition and, therefore, the function returns `False`.

Exercise 9: 2D Most Rewarding Shortest Path

Complete the function `rewardShortPath(env)`, which finds the shortest path from a starting cell A to an arrival cell B on a grid, having the highest reward.

Attribute `env` describes the environment. An environment is a matrix that can contain the following characters: A, B, O, X, and R. A and B are the starting and arrival cell, respectively. An O represents an empty space, an X represents an obstacle, and an R represents a reward. An example is presented in Table 2.

A	X	R	R	R
O	O	O	X	B
O	O	O	O	R

Table 2: Sample 2D environment.

A cell is adjacent only to other cells in the same row or column. For example, the cell containing an A in the top-left corner is adjacent only to the X on the left and the O at the bottom - diagonals are not considered adjacent.

When searching for the shortest path, only empty and reward cells (Os and Rs, respectively) can be traversed. If a path passes through a reward cell, the reward is collected. Figure 1 shows two shortest paths (note: there are actually more than two shortest paths in this example), one approaching the B from the top (in red) and the other from the bottom (in blue); both

have a length of seven. However, the former has a total reward of three, while the latter has a total reward of one. Therefore, the red path is the one that the function is looking for.

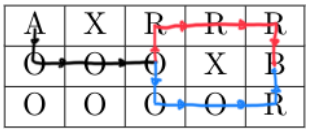

Figure 1: Two shortest paths.

Parameter `env` is given to the function as a list of sublists in Python, or an Array of arrays in JavaScript, where each sublist/array contains characters corresponding to a row. The representation corresponding to the environment above would be:
`[['A','X','R','R','R'],['O','O','O','X','B'],['O','O','O','O','R']]`.

The function should return a 2-value tuple in Python, or a 2-value Array in JavaScript containing the length of the shortest path and the total reward collected.

Example: Given the environment above, the function should return `(7,3)`.

Hint: You might want to enumerate all the paths from A to B (excluding those that pass through the same cell twice). Then, you can can choose the shortest path having the highest reward.

Exercise 10: Social Network Analysis

Social network analysis (SNA) is the process of investigating social structures through the use of networks and graph theory. It characterizes networked structures in terms of nodes (individual actors, people, or things within the network) and the ties, edges, or links (relationships or interactions) that connect them (Wikipedia contributors, 2021e). Figure 2 shows a sample social network with seven actors.

Network (and graphs) can be represented as adjacency matrices. An adjacency matrix is a square matrix having as many rows and columns as there are actors in the network. A is 1 if the actors corresponding to the row and the column are connected by an edge, and is 0 otherwise. The matrix corresponding to the social network in Figure 2 can be found below in Table 3. In turn, matrices can be represented as a list of sublists, as explained in Exercise 9.

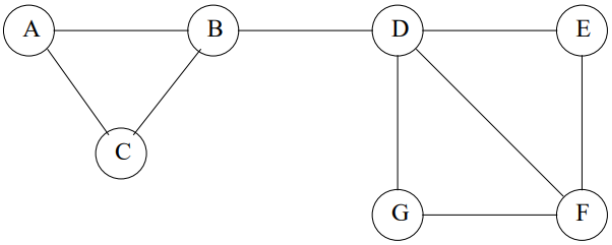

Figure 2: Sample social network with 7 actors.

	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	1	0	1	1	0	0	0
C	1	1	0	0	0	0	0
D	0	1	0	0	1	1	1
E	0	0	0	1	0	1	0
F	0	0	0	1	1	0	1
G	0	0	0	1	0	1	0

Table 3: Adjacency matrix corresponding to the social network in Figure 2.

In graph theory, a clique is a subset of nodes of a graph such that every two distinct nodes in the clique are adjacent; that is, every pair of nodes in the clique must be connected. A maximal clique is a clique that cannot be extended by including one more adjacent vertex (Wikipedia contributors, 2021a). In SNA, a clique represents a tightly-knit group or community. Figure 3 illustrates all the maximal cliques that can be found in the sample network: (A,B,C), (B,D), (D,F,G), and (D,E,F). Following the definition above, node E cannot be added to the yellow click (D,F,G), as node E is not connected to node G and, therefore, (D,E,F,G) would not be a clique.

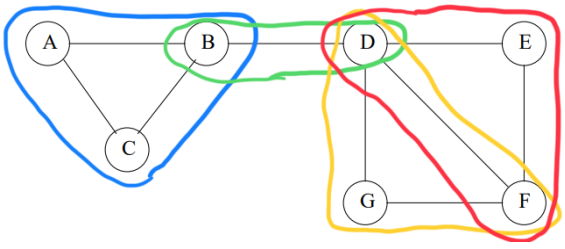

Figure 3: Maximal cliques in the sample social network. Each clique is highlighted in a different colour.

As it can be seen from the example, a node can belong to multiple maximal cliques. It is interesting to identify actors that belong to multiple cliques, as they can act as bridges between communities and be extremely influential.

In this exercise you need to complete the function `cliqueCounter(network)` which, given the adjacency matrix of a social network as a list of sublists or an Array of arrays, returns the number of maximal cliques that each actor belongs to as a list.

Example: The result of applying `clickCounter(network)` on the adjacency matrix in Table 3 would be `[1,2,1,3,1,2,1]`.

Criteria for assessment

Each exercise can be awarded a maximum of 10 marks. Therefore, perfectly solving five exercises will give you 50 marks (pass), and perfectly solving ten exercises will give you 100 marks.

Each exercise is marked for both function (8 marks max) and style/quality (2 marks max). Exercises that are not a real attempt at solving the problem presented will receive zero marks.

Functionality [8 marks/exercise max] The functional part of the submission is automatically marked by scripts that run the completed function against a set of test cases. You will be provided with two test cases per exercise. During marking, each exercise will be tested against four test cases, including the two provided. For each language (Python and JavaScript) and exercise, passing one test will award you 1 mark. Therefore, the maximum functionality mark of 8 will be given only if all the tests are passed in both languages.

Code quality and style [2 marks/exercise max] Each version of the exercise (i.e., Python and JavaScript) is assessed independently, according to the following criteria:

High quality and style (50-100%, 0.5-1 marks per exercise)	Low quality and style (0-50%, 0-0.5 marks per exercise)
Code is elegant	Code is messy or overly verbose
Code has no redundancies	Code has multiple redundancies and repetitions
Code is well commented	Code is lacking in meaningful comments
Code is perfectly modular (i.e., appropriate functions and/or classes defined)	Code is disorganised
Code makes smart use of built-in language features and classes.	Code does not make use of language features

Therefore, an exercise solved in only one language could achieve at most one of the style/quality marks. To obtain both marks, the exercise must be solved in both languages. Also, only exercises that pass at least two tests are evaluated for style and quality.

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned on the return date via Learning Central and/or email.

The feedback from this assignment will be useful for your second programming assignment, and will also be relevant for any future programming tasks.

References

Wikipedia contributors. (2021a). *Clique (graph theory) — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Clique_(graph_theory)` ([Online; accessed November 2, 2021])

Wikipedia contributors. (2021b). *Comma-separated values — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Comma-separated_values` ([Online; accessed November 2, 2021])

Wikipedia contributors. (2021c). *Natural language processing — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Natural_language_processing` ([Online; accessed November 2, 2021])

Wikipedia contributors. (2021d). *Rock paper scissors — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Rock_paper_scissors` ([Online; accessed November 2, 2021])

Wikipedia contributors. (2021e). *Social network analysis — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/wiki/Social_network_analysis` ([Online; accessed November 2, 2021])