Module Code
COM00015C

BEng, BSc, MEng and MMath Degree Examinations 2023–2024

Department Computer Science

Title Software 1: Formative Assessment 2

Time Allowed FIVE hours

Papers late by up to 30 minutes will be subject to a 5 mark penalty; papers later than 30 minutes will receive 0 marks.

The time allowed includes the time to download the paper and to upload the answers.

Word Limit None

Allocation of Marks: Questions 1, 2, 3, and 4 are worth 10 marks, and question 5 is worth 40%. The remaining 20% are for code documentation (docstring & comments), code clarity and code style. The code must adhere to the Google Style Guide and PEP0008. All questions are independent and can be answered in any order.

Instructions:

Candidates must answer **all** questions using Python 3.10 or above. Failing to do so will result in a mark of 0%. Download the paper and the required source files from the VLE, in the "Assessment> Formative Assessments 2" section. Once downloaded, unzip the file. You **must** save all your code in the files provided. **Do not** save your code anywhere else other than this folder.

Submit your answers to the GradeScope submission point named **SOF1 2023-24 formative assessment 2**. You can find the submission point on the "Assessment>Formative Assessment 2" page on the VLE.

Note on Academic Integrity

We are treating this online examination as a time-limited open assessment, and you are therefore permitted to refer to written and online materials to aid you in your answers. However, you must ensure that the work you submit is entirely your own, and for the whole time the assessment is live you must not:

- communicate with other students on the topic of this assessment.
- communicate with departmental staff on the topic of the assessment other than to highlight an error or issue with the assessment which needs amendment or clarification).
- seek assistance with the assessment from academic support services, such as the Writing and Language Skills Centre or Maths Skills Centre, or from Disability Services (unless you have been recommended an exam support worker in a Student Support Plan).
- seek advice or contribution from any other third party, including proofreaders, friends, or family members.

We expect, and trust, that all our students will seek to maintain the integrity of the assessment, and of their award, through ensuring that these instructions are strictly followed. Where evidence of academic misconduct is evident this will be addressed in line with the Academic Misconduct Policy and if proven be penalised in line with the appropriate penalty table. Given the nature of these assessments, any collusion identified will normally be treated as cheating/breach of assessment regulations and penalised using the appropriate penalty table (see AM3.3. of the Guide to Assessment).

1 (10 marks) Basic Programming Structure

The code must be written in the provided file `question_1.py`.

The scenario for this question is about computing the score of an athlete in a given track event. We need to convert a time in seconds into points. The formula is:

$$points = a(b - time)^c \tag{1}$$

where $time$ is the time in seconds of the athlete for that event. a, b and c are parameters that vary depending on the event (see Table 1). The value of points must be rounded down to a whole number after applying the respective formula (e.g. 499.999 points becomes 499). If the value of points is less than 0, then 0 should be returned instead.

Women's events	a	b	c
200m	4.99087	42.5	1.81
800m	0.11193	254.0	1.88
110m	9.23076	26.7	1.835

Table 1: Constants a, b and c for each event.

Write a function `track_points(time, eventParameters)` which takes a `float` parameter `time` representing the athlete's time in seconds, and a tuple containing the event's parameters (a, b, c) in that order. The method returns an `int` representing the points scored for that event using Equation 1. The method raises a `ValueError` if `eventParameters` does not have exactly 3 values.

For example:

- 200 metres time of 22.83 seconds corresponds to 1,096 points
- 110 metres hurdles time of 12.54 seconds corresponds to 1,195 points
- 800 metres time of 128.65 seconds (i.e. 02:08.65) corresponds to 984 points

2 (10 marks) List Built-in Data Structure

The code must be written in the provided file `question_2.py`.

Write a function `common_values(lst1, lst2)` which returns the set of values that are present in both lists `lst1` and `lst2`. If there are no common values, the function should return an empty list. In addition, the returned list should not contain any duplicates. Note also the order of the elements in the returned list does not matter.

For example:

```
>>> common_values([1,1,2,3,4,5,6], [9,3,1,7,5,1])
[1,3,5]
>>> common_values([2, 4, 6, 8], [1, 3, 5, 7, 9])
[]
```

3 (10 marks) Dictionary and I/O

The code must be written in the provided file `question_3.py`.

Write a function `read_from_file(filename)` that takes the pathname of a text file as parameter and returns the content of the file into a dictionary.

The text file contains the time of athletes competing in a Women's heptathlon event. However, for simplicity we will be considering only three events out of the seven comprised in an heptathlon. The format of the text file is as follow:

- Lines starting with # are comments and should be ignored.
- All other lines represent the data for a single athlete. The fields recorded on each line are the athlete's name, her 200m split time in seconds (that is the time recorded on the 200 metres race), 110m split time in seconds, and 800m split time in seconds. The fields are separated by a comma.

The returned data has the following format:

- the data is a dictionary where the keys are athlete's names and the values are dictionaries containing the split times for that athlete.
- the dictionary containing the split times has the category of the recorded time as keys (i.e. `"200m"`, `"800m"`, and `"110m"`), and the values are the recorded time in seconds for that category (as `float`, not string).
- for example:
  `{'Ada Boole':{'200m':24.25,'800m':143.01,'110m':13.50},...}`

The function must raise a `ValueError` if the file does not respect this format.

4 (10 marks) Recursion

The code for this question must be written in the provided file `question_4.py`.

Warning: if the implemented function for this question is not recursive, a mark of 0 will be awarded. The aim of the exercise is to check if a game level is feasible or not. The game is a simple 2D platform composed of springboards with power ups and deadly traps (represented by mines). The power ups' number below the springboard represents the player's maximum jump length from that board. The aim of the game is to depart from the first springboard and arrive to the last one without stepping on a mine. Figure 1 shows a game level that is feasible, whereas Figure 2 shows a game level that a player cannot win.

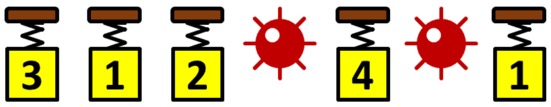

Figure 1: Example of a game level that is feasible. One solution is to jump 2 steps from index 0 to 2, then 2 steps from index 2 to 4 and finally another 2 steps to the last index.

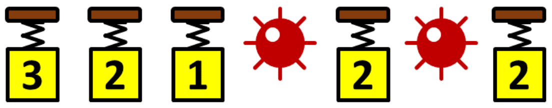

Figure 2: Example of a game level that is impossible to complete. Whatever the choice you make, you will always arrive at index 3 which is a deadly trap. It is therefore impossible to reach the last index.

To represent the level, we have chosen a list of non-negative integers. The starting point of the level is at the first index of the list, and the exit is at the last index of the list. The numbers in the list represent the power up of the springboard at that position. A mine is represented by the value 0. In this context, if the element at the last index is 0 then the level is not feasible. For example, the board shown in Figure 1 is represented by the list `[3,1,2,0,4,0,1]`. Similarly, the board shown in Figure 2 is represented by the list `[3,2,1,0,2,0,2]`.

(i) [10 marks] Implement a **recursive** function `check_level(level)` that takes a list of positive integers representing a game level and returns `True` if the level is feasible, `False` otherwise. The efficiency of the solution does not matter, you should implement a brute force approach that tries all possible cases.

5 (40 marks) User Defined Data Structures

The code for this question must be written in the provided file `question_5.py`.

In this question, we will start building the game Blotris, a knock-off version of Tetris™. The aim is to add many random shapes to the board before we cannot add more. In Figure 3 we can see how a shape is added at a given position. When a row is fully occupied, it is removed and the rows above that row are shifted down. For example, when adding the shape in Figure 3, two rows are fully occupied. Figure 4 shows how these two rows are removed and how the rows above them are shifted down.

(i) [10 marks] Implement a class `Blotris`, with a constructor that takes two integer parameters `rows` and `cols` in that order. The parameter `rows` represents the number of rows on the board, and `cols` the number of columns. The board shown in Figure 3 was constructed with 8 rows and 5 columns.

- the class `Blotris` has an instance attribute `_board` which is a 2D list of integers representing the board. The value 0 represents an empty block, the value 1 an occupied block.
- the constructor should initialise `_board` to a 2D list of `rows` rows and `cols` columns containing only zeros.
- The constructor raises a ValueError if `rows` or `cols` is not greater or equal to 5.
- You can add more instance attributes if needed.

(ii) [15 marks] Implement the method `add(self, shape, row, col)` that adds a shape to a game board at row `row` and column `col`. The pair (row, col) is where the top-left corner of the shape should be. Figure 3 shows a shape added at $row = 3$ and $col = 1$. Remember rows and columns start at index 0.

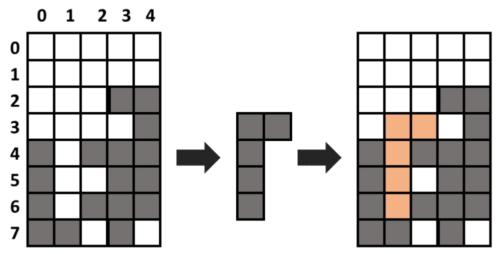

Figure 3: Example of adding a shape to the game board at row 3 column 1, that is the fourth row and the second column.

A shape is represented by a 2D list, where 0 represents an empty block and 1 an occupied block.

For example, the shape added in figure 3 is represented by:

$$[[1,1],[1,0],[1,0],[1,0]]$$

- A shape can be added at a given position on the board only if it does not overlap an occupied block and is within the bounds of the board. For example, the shape in Figure 3 could not be added at $row = 3$ and $col = 0$.
- If a shape can be added, then the board is changed accordingly and the method `add` returns `True`. Otherwise, the method returns `False` and the board remains unchanged.
- If the shape is out of the boundary of the board, the method returns `False` but **must not** raise an exception.

(iii) [15 marks] Implement the method `update(self)` that updates the board, that is removes fully occupied rows and shift down the rows above accordingly. Figure 4 shows an example. In this example, rows 4 and 6 must be removed (red marks), and all rows above shifted accordingly. It can be done in several steps, first by removing the full row closest to the bottom and shifting the rows above, then removing the next full row closest to the bottom and shifting the rows above, and so on until there are no more full rows on the board. Using this algorithm, some rows may be shifted multiple times. If you prefer you can implement your own algorithm to solve the problem. It should be noted the size of the board **must not** change. The method does not return anything.

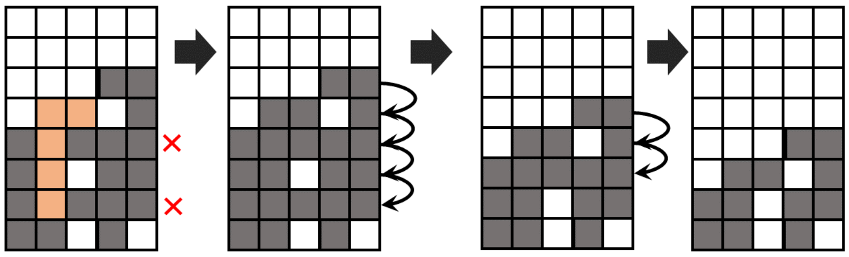

Figure 4: Example of updating the board to remove rows that are full.

End of examination paper