Cardiff School of Computer Science and Informatics

Coursework Assessment Pro-forma

Module Code:	CM1103
Module Title:	Problem Solving with Python
Lecturer:	Stuart Allen
Assessment Title:	Problem solving exercise
Date Set:	27th November 2018
Submission date and Time:	14th December 2018 at 9:30am
Return Date:	11th January 2019

This coursework is worth 40% of the total marks available for this module. The penalty for late or non-submission is an award of zero marks.

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

Description		**Type**	**Name**
Cover sheet	Compulsory	One pdf file	`[Student number].pdf`
Answer to Q1	Compulsory	One Python file	`Q1_[Student number].py`
Report for Q2	Compulsory	One Word or pdf file	`Report_[Student number].doc/docx/pdf`
Source code for Q2	Compulsory	One or more Python files (may be contained in a single zip file)	No restriction

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

Assignment

This coursework asks you to investigate the fairness of a scoring systems for sailing competitions composed of n single handed yachts which compete over a series of 6 races. All sailors start each race at the same time, with the first to finish awarded 1 point, the second awarded 2 points, the third 3 points and so on. After the series is complete, each sailor discards their *worst* score, and totals the rest.

For example, consider the following results:

Name	Race 1	Race 2	Race 3	Race 4	Race 5	Race 6
Alice	1	2	1	1	1	1
Bob	3	1	5	3	2	5
Clare	2	3	2	2	4	2
Dennis	5	4	4	4	3	4
Eva	4	5	3	5	5	3

Their total scores for the series would be:

Name	Discard worst
Alice	5
Bob	14
Clare	11
Dennis	19
Eva	20

Modelling sailors ability

We are interested in how the scoring system rewards skill and consistency. To enable us to simulate races, assume each sailor is defined by a *mean skill* value μ and *standard deviation* σ. For a given race, we will randomly select a *performance score* for each sailor using the *normal distribution*. The order in which sailors complete the race is then given by their respective performance scores, with the highest score winning the race.

Questions

1. Provide code to answer the questions below. There will be an additional mark covering the quality and style of your code.

   [Code quality: 5 marks]

   (a) We will store the results for a single sailor over a series as a tuple containing their name and a list of their finishing positions. For example:

   ```
      ("bob", [2, 4, 1, 1, 2, 5])
   ```

   Write a function named `series_score` that takes a sailor's results as a **single** argument, and returns their score for the series as the sum of all positions except the worst.

   Passing the data above to the function should return ***`10`****.*

   [Functionality: 3 marks]

   The function should allow the number of races to be discarded from the series to be passed as an optional second parameter.

   Passing the data above to the function, along with 2 should discard the worst two races and return ***`6`****.*

   [Functionality: 2 marks]

   (b) Write a function named `sort_series` which takes a list of sailor series results as a **single** argument (e.g

   ```
   [("Alice", [1, 2, 1, 1, 1, 1]), ("Bob",   [3, 1, 5, 3, 2, 5]),
      ("Clare", [2, 3, 2, 2, 4, 2]), ("Dennis", [5, 4, 4, 4, 3, 4]),
      ("Eva", [4, 5, 3, 5, 5, 3])]
   ```

) and returns a **copy** of the list sorted in ascending order of their series score (break ties using their position in the first race).

   Passing the data above to the function should return: ***`[('Alice', [1, 2, 1, 1, 1, 1]), ('Clare', [2, 3, 2, 2, 4, 2]), ('Bob', [3, 1, 5, 3, 2, 5]), ('Dennis', [5, 4, 4, 4, 3, 4]), ('Eva', [4, 5, 3, 5, 5, 3])]`****.*

   [Functionality: 4 marks]

   (c) Assume sailors performances are given in a csv file with the format:

   ```
   name, mean performance, std dev
   Alice, 100, 0,
   Bob, 100, 5,
   Clare, 100, 10,
   Dennis, 90, 0,
   Eva, 90, 5,
   ```

   where each record specifies their name, mean performance μ and standard deviation σ. Write a function named `read_sailor_data` that reads in a csv file of this format and returns a dictionary with names as keys whose values are performance and standard deviation pairs.

   Reading in the data above should return: ***`{'Dennis': (90.0, 0.0), 'Clare': (100.0, 10.0), 'Eva': (90.0, 5.0), 'Bob': (100.0, 5.0), 'Alice': (100.0, 0.0)}`*** *(note the order of items may vary).*

   [Functionality: 3 marks]

The above hint may give different results since dictionaries in Python are not guaranteed to store the key, value pairs in the same order as they were added. To allow testing the data, you could instead return an `OrderedDict` instead, which doesn't have this restriction. To do this, add `from collections import OrderedDict` to the start of your file, and create a dictionary using `d = OrderedDict()` instead of `d = {}`. Reading in the data above should then return: `OrderedDict([('Alice', (100.0, 0.0)), ('Bob', (100.0, 5.0)), ('Clare', (100.0, 10.0)), ('Dennis', ((90.0, 0.0),)), ('Eva', (90.0, 5.0))])`.

(d) Write a function named `generate_performances` that takes a dictionary of the format returned by your answer to 1c as a **single** argument, generates a random performance value using the normal distribution for each sailor, and returns these in another dictionary with names as keys.

Using the data returned by the example in 1c should return: `{'Dennis': 90.0, 'Alice': 100.0, 'Bob': 101.4389222493041, 'Eva': 94.18226076274071, 'Clare': 111.52090179040226}` (note the order of items may vary; exact results will depend on the random seed – 57 was used here).

[Functionality: 3 marks]

Using the OrderedDict version avove, this should return: `{'Alice': 100.0, 'Bob': 105.76045089520113, 'Clare': 108.36452152548142, 'Dennis': 90.0, 'Eva': 96.10844089749128}`.

(e) Write a function named `calculate_finishing_order` that takes a dictionary of the format returned by your answer to 1d as a **single** argument, and returns a list of each sailor's position in the race.

Using the data returned by the example in 1d should return: `['Clare', 'Bob', 'Alice', 'Eva', 'Dennis']`.

[Functionality: 4 marks]

(f) Create a dictionary `results` with names as the keys, and empty lists as the values. Using the functions above, run 6 races, appending the positions of each sailor to the corresponding list, calculate their series score and output their names in order.

Using the data from previous questions and 57 as the random seed should give: `{'Dennis': [4, 4, 4, 4, 4, 5], 'Alice': [2, 2, 2, 1, 1, 2], 'Bob': [3, 3, 3, 3, 2, 1], 'Eva': [5, 5, 5, 5, 3, 3], 'Clare': [1, 1, 1, 2, 5, 4]}` giving the order `['Alice', 'Clare', 'Bob', 'Dennis', 'Eva']`.

[Functionality: 6 marks]

2. Produce a **brief** scientific report using the supplied template (typical length should be one side of A4 – maximum allowed length is two sides of A4) that uses the results of simulation (with suitable input data) to investigate whether:

   - Is it better to be a consistent or inconsistent sailor under this scoring scheme?
   - Does this depend on how skilful you are relative to other sailors?
   - How would this change if either no races were discarded or two races were discarded?

   Your answer should include at least one relevant figure generated with `matplotlib`.

   You should include a separate file containing the Python code used to generate your results and figure.

   [Report: 5 marks; Achievement: 5 marks]

Learning Outcomes Assessed

- Use Python and common modules to implement simple algorithms expressed in pseudocode, and understand fundamental programming concepts
- Develop informal algorithms and apply recursion to solve simple problems
- Write scientific reports describing the analysis of a problem

---

Criteria for assessment

Functionality: Does the code perform the required task correctly and accurately (for the given test data and other reasonable inputs[1])?

Code quality: Is the code elegant and well-written; easy to run; simplified by the use of built-in languages features where appropriate; readable and easy to follow? Are appropriate functions defined and written to enable reuse between questions?

Report:

4–5 marks: Relevant content covering all required elements; excellent evidence supporting conclusions; accurate; clear and concise description; clearly labelled figure.

2–3 marks: Relevant content covering many of the required elements; good evidence; few errors/omissions; generally clear; some omissions in figure.

0–1 marks: Little or no relevant content; lack of evidence; extensive errors/omissions; unclear description; no figure.

Achievement:

4–5 marks: Fully addresses scope of question.

2–3 marks: Broadly addresses scope, with some omissions or errors.

0–1 marks: Little or no achievement; many significant errors.

Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned in Week 12 via a model solution (uploaded to Learning Central), an email with marks and individual feedback, and a drop-in session.

Feedback from this assignment will be useful for any subsequent modules involving programming, including Developing Quality Software (CM1202), Object Oriented Java Programming (CM1210) in year 1, and Object Orientation, Algorithms and Data Structures (CM2307), Data Processing and Visualisation (CM2105) in year 2.

[1] You may assume arguments of the correct type are given, and input files have the correct format.