Coding Challenge

Solving programming challenges is a fundamental skill within computer science. When developing complex programs there are often many challenges to solve both when implementing features or when performing the various other tasks required for successful software development. In the real world, you cannot be expected to know how to do everything off the top of your head, therefore the ability to efficiently research is also a key skill in being an effective software developer.

Context

As we all know sailing is one of the most historic and best sports to exist. Something equally as prestigious is the Olympic games which takes place every 4 years. Next year this happens in Paris, France but the organisers need help calculating the score for the sailing. This is your job!
Below is a series of tasks for you to complete, each task somewhat builds on from the last and vary in difficulty. Here you will solve the tasks within Python by applying the skills you have learned within this course unit and some individual research.

In our scenario there are a total of 10 countries participating in the Olympic sailing event. Each country is given a unique number from 01 to 10. There are also 10 types of boats being raced, again each being given a unique number from 01 to 10. Each country is represented exactly once in each type of boat. There are two different types of races. Race 01 is a single points race and Race 02 is double points. Points are awarded to the sailor's respective placing in a particular race, e.g. if a boat finishes 1^{st} then they receive 1 point, 2^{nd} gets 2pts and so on until 10^{th} gets 10pts. The winner of a racing series is the country with the lowest points total.

Input File Structure

```
0201-0701-0602-0503-0104-0905-0306-0807-0408-0209-0210
0602-0201-0402-0603-0504-0905-0306-0707-0108-0809-1010
0501-0201-0702-01xx-0804-1005-0306-0907-0508-0409-0610
0701-0301-0902-0103-0604-0205-0406-0707-0508-1009-0810
```

In the above structure we will use the first line as an example. The first 4 digits (i.e. `0201`) represent the Type of Boat and the Type of race taking place. So `0201` mean Boat Type 02 and Single Points Race. The subsequent remaining digits represent the country and their position in the race. So, country 07 finished 1^{st}, country 06 finished 2^{nd}, country 05 finished 3^{rd}, and so on, until you reach country 02 which has finished 10^{th}.

On the next line the `0602` would mean Boat Type 06 and double points race, followed by the countries and their respective finishing position within that race. A double points race means that a boat finishing 1^{st} gets 2 points, 2^{nd} get 4 points and so on.

There may be an occurrence where you see a country's finishing position is xx such as country 01 below:

```
0501-0201-0702-01xx-0804-1005-0306-0907-0508-0409-0610
```

This means that this country has been disqualified from the race and will receive 11 points. Every country which finished behind this will now move up one place, i.e. country 08 will now finished 3rd instead of 4th, country 10 4th in stead of 5th, etc. If there are multiple boats disqualified from a race, all disqualified boats receive 11 points irrespective of their original position in the race and will appear in the original order in any outputs. Again, any countries after the disqualified country will move up a position. In a double points race any disqualified boats receive 22 points.

For example, `0601-10xx-0102-0503-0304-02xx-0606-0807-0708-0409-0910` means, boat type 06, single points. The points of this race would be:

Country	01	05	03	06	08	07	04	09	10	02
Points	1	2	3	4	5	6	7	8	11	11

Tasks

In this work, there are 5 tasks to complete. Each of these tasks is associated with a method within the provided .py file however there are dependencies within this work. These tasks are each part of a process which should be followed in order, examples of expected output formatting are given for each section.

Section 1: File Input

(a) Read and store all of the race results into a list of strings which for this specification we will call the "results string"

(b) Each race in the input.txt file is on a new line

(c) Return the results string

Section 1 Output Format:

```
['0902-0701-0302-0403-08xx-0605-1006-0207-0508-0909-0910',
'0201-0701-0602-0503-0104-0905-0306-0807-0408-0209-0210',
'0402-0801-0902-1003-0604-0305-0406-0107-0208-0509-0510',
'0401-0601-1002-0703-0104-0405-0806-0307-0208-0909-0910',
'0101-1001-0802-0403-0704-0105-0606-0307-0908-0209-0210']
```

Section 2: Specific Race Results

(a) Output the results of Boat 'n' Race 'x' where n & x are give by the automarker

(b) The race number for a boat is the order in which they are read from the input file. i.e. the first occurrence of boat type 01 is their first race, the second occurrence is their second race, and so on.

(c) Your program should only return the race results and not the boat type. e.g. `0401-0601-1002-0703-0104-0405-0806-0307-0208-0909-0910` would become `0601-1002-0703-0104-0405-0806-0307-0208-0909-0910`

(d) If there are not enough races for that Boat then return an empty string. E.g. if Boat = 02 and Race = 04, but Boat 02 only has 03 races then return " "

(e) Each Boat Type can have any number of races.

(f) Return the race results string

Section 2 Output Format:

"`0601-1002-0703-0104-0405-08xx-0307-0208-0909-0910`"

Section 3: Boat Type Results Table

(a) Output the overall results for boat 'x'. x is provided by the automarker

(b) Calculate the total points scored by each country for boat type 'x'

(c) These should appear in accessing order with the winning boat (i.e. lowest points total) appearing first

(d) Tie breakers:

  (i) If there is a tie on points the boat that performed best in the last race of that boat type is ahead on the leaderboard

  (ii) If there are are multiple countries disqualified from the final race then they will go in their order before they were disqualified, e.g.
  `0202-0701-0502-04xx-0204-0105-03xx-0907-1008-0609-0810`
  would be placed into the following order:
  `07, 05, 02, 01, 09, 10, 06, 08, 04, 03`.

(e) Output format is "CountryNumber-PositionInTable-TotalPoints" e.g. `05-01-25`

(f) Return the class table string

Section 3 Output Format:

"`05-01-25, 03-02-36, 06-03-41, 08-04-43, 04-05-44, 09-06-45, 07-07-48, 10-08-50, 01-09-53, 02-10-58`"

Section 4: Results Tables inc. Discards

(a) Each individual boat can Discard their worst result from each race type and exclude them from their total score

(b) Each racer will discard their worst single point race and their worst double points race

(c) If there are 2 or less races of a certain race type then all sailors count all of that type of race's scores and no results are removed for that boat and race type

(d) If a sailor has multiple races with the same worst score remove the last instance of it

(e) A sailor discarding a result does not affect the results of other sailors in that race

(f) Output format is "CountryNumber-PositionInTable-TotalPoints" e.g. `05-01-25`

(g) Return the class table discard string

Section 4 Output Format:

"`05-01-21, 03-02-27, 06-03-33, 04-04-34, 08-05-36, 09-06-37, 07-07-38, 10-08-39, 02-09-47, 01-10-47`"

Section 5: Overall Country Medal Table

(a) Determine how many medals each country wins. If a country finishes in the top 3 of a certain boat type then they will win the respective medal for that boat

(b) Positionings are based on the results tables from section 4

(c) A Gold meal is worth 3pts in the Medal table, Silver = 2pts, Bronze = 1pt

(d) In the case of ties the display order is total medals score, then golds, silvers, bronze, country number

(e) Output format is "Country-Golds-Silvers-Bronzes-TotalMedalsScore" e.g. `03-02-04-01-15` or `05-00-01-00-02`

(f) Return the medal table string

Section 5 Output Format:

```
"06-01-03-02-11, 03-01-03-00-09, 05-02-01-00-08, 09-02-00-01-07, 08-01-01-02-07,
07-02-00-00-06, 01-00-01-01-03, 10-00-00-02-02, 04-00-00-01-01, 02-00-00-00-00"
```

Provided Materials

For this work we provide you with two files:

- A Python file called `race_solution.py`
- An example input file called `input.txt`

Important Notes:

1. The input file should always be in the same directory as the python file *(and not contained in any other folder)*
2. All instructions within the Python file should be followed exactly
3. The text file will always be named "input.txt"
4. Import statements are **not** allowed for this task
5. There is no required output for this work. All output is done via method return statements. Methods will be called directly. Print statements may cause the tests to fail to so avoid them or remove them before submission.
6. You are only permitted to submit a single python file for this work (do not submit test files or any other files). Do not rename the python file.
7. You do not need to submit the "input.txt" file
8. Do not you use absolute file paths or the auto-marking will fail with **no** exceptions for this

Preparation and Submission

Your `race_solution.py` file must be pushed to your Gitlab (*16321_python_work_[username]*) remote repository, it must have the tag *Coding-Challenge* and be on a branch named *Cwk-03*. Failure to correctly tag your work or placing it on an incorrect branch may cause the marking to fail resulting in a mark of 0.

The code will be autograded using a system called GradeScope. Gradescope will pull code from your gitlab repository and then run a sequence of tests, generating a mark. The outcome of some of these will be visible to you immediately, the rest remain hidden until the results are published. To access Gradescope you will need to click on the tab "Gradescope" on the left hand side of the COMP16321 Blackboard page.

On the submission page for the exercise, you will see an option to submit via gitlab. When you do this for the first time, you will be asked to authorise the application. Once you have done this, you will be able to select a repository and branch to submit. The code on that branch will then be pulled by Gradescope and tests will be run.

Note that you will receive some feedback on test performance, but your mark will not be visible until after they have been published. You can continue to develop code and resubmit as many times as you wish before the deadline: tests will be rerun on re-submission. Do **not** use any non-standard libraries (e.g. libraries that must be installed via pip). If you do this, the code may fail to run.

The test scripts will not be expecting any additional files other than `race_solution.py`. If your code relies on other files (with the exception of the input file which does not need to be submitted), it may fail to run.

If your code does not run, this will be reflected in the feedback for the initial tests. Code that does not run will attract a low mark. It is **your** responsibility to check that the autograder has been able to run your code. Making amendments to the file you were provided may cause the marking to fail and you will not receive the marks for the affected portion.

The submission slot on Gradescope will close at exactly 18:00 on the 8th December. It is your responsibility to ensure that your work is submitted on time, we will not make exceptions for submitting late. If you miss the submission cutoff then you will receive a mark of 0. There are **NO** extensions available for this work (*this includes DASS auto extensions*).

Deadline:

18:00 on Friday the 8th December 2023

Assessment Type:

If the marking system finds that the file submitted is unreadable and/or does not compile then you will not receive any marks for that program. Please be aware that we will not install external packages to make your code compile. We should be able to run your code with a basic python setup.

Please note: Your work will not be re-marked because you disagree with the mark you were awarded. This work has been run through the same automarking system as all other students. Re-running the automarker will not output a different result. Any queries with your marking can be raised with either Gareth, Stewart or Terry within a week of the marks being released, any queries made after this time will not be considered.

End of Assessment