Algorithms & Data Structures

I/O and Error Handling

To date we have use variables to store set of data, however when the program exits, all data is lost. There is a need for persistent data. To alleviate the problem, we use files to store the data persistently. There are several types of files, binary files, random access files, but the simplest to use are text files and our focus will be only on text files. You could read more about text files in Python on the [course online book](https://drlilianblot.gitbook.io/introduction-to-programming-with-python/file-i-o).

**Write all your code in a file named** week6exercises.py.

## Exercise 1:

**Note: Read the entire function requirements before starting the implementation.**

We want to create a function sum\_from\_file(filename) that calculate the sum of all int contained in the text file filename. The format of the text file is as follow, a series of int separated by a space spanning several lines as shown below. In the example below the returned value should be 100.

1 30 4 5

8 12 19 1

5 5 10

1. It is sometime useful to decompose the problem into smaller problem. In this case it would be useful to have a function sum\_numbers(a\_string) that calculates and returns the sum of all numbers contained in the string a\_string. The format of the string is a series of int separated by a space. For example:

>>> sum\_numbers('1 30 4 5')

40

It could be useful to remember /check some the methods already existing for str object.

1. The function should raise a ValueError when the format of the file is not as described.
2. The function should return None if the file passed in parameters does not exist.
3. Write the docstring (python documentation) for this function, as explain in [chapter 9 of the textbook](https://drlilianblot.gitbook.io/introduction-to-programming-with-python/code-documentation/documenting-code-via-python-docstring).

## Exercise 2:

The aim of this exercise is to compute the score of an athlete in a given track event. We need to convert a time in seconds into points. The formula is:

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.

Table : Constants a, b and c for each event

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

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 provided earlier.   
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

## Exercise 3:

Write a function rasterise(list\_1D, width) that transforms a 1D list passed as parameter into a 2D list, where each sub-list have width elements. If the length of the 1D list is not a multiple of width, the function must raise a BufferError with an appropriate error message. If width is less than 1, the function must raise a ValueError with an appropriate error message.

For example:

>>> rasterise([1,2,3,4,5,6,7,8],4)

[[1,2,3,4],[5,6,7,8]]

>>> rasterise([1,2,3,4,5,6,7,8],2)

[[1,2],[3,4],[5,6],[7,8]]

>>> rasterise([1,2,3,4,5,6,7,8],3)

BufferError: invalid width!