ECM1400 Programming
Continuous Assessment

Date set:	4th November, 2022
Hand-in date:	**11:59am 16th December, 2022**

This continuous assessment (CA) comprises 100% of the module assessment.

Note that electronic submission is required through BART.

Version: **1.0**
Last update: **4 of November 2022**

Contents

1 General instructions

- This CA is designed to test the programming concepts that will be covered in the entire module.
- This assignment will require the application of fundamental programming constructs, control flow, data structures and design patterns to develop and deliver a software product to solve a real-world problem.
- Note that some of the content is only taught within 2 weeks of the deadline, however, this should not stop you from making a start and implementing a lot of the functionality well in advance of the deadline.
- You should carefully follow the functionality described.
- Note some requirements intentionally do not include some details and you should make design decisions carefully.
- The CA requires electronic submission to the BART online submission platform.
- You should submit your Python code and other outputs via the electronic submission system at `https://bart.exeter.ac.uk/` under CEMPS and Harrison. Use the category containing ECM1400 and 2022-23, Continuous Assessment.
- Upload a compressed version of your files as a single file using the `zip` compression format.
- You should upload your file ahead in advance of the deadline to avoid a late cap due to Internet problems or other technical difficulties.
- No extension will be granted on the grounds of technical difficulties.
- Sending the file via e-mail is NOT considered a submission.
- It is your responsibility to make sure you submitted the correct file. No extension will be granted on the grounds of mistakes in the upload process (e.g., the wrong version, a PDF, a link to the file etc.).
- The understanding of the problem specifications and requirements described in this document is part of the assessment process.

2 Introduction

The environmental, economical and societal challenges imposed by the ongoing climate crisis are already imposing significant challenges on governments worldwide. In 2019, carbon emissions originating from road transport (111 $MtCO_2e$) accounted for more than 24% of the UK's total greenhouse gas emissions. From the total transport-related emissions, 61% (68 $MtCO_2e$) were produced by cars and taxis.

As part of the NetZero commitments assumed by the UK government, you were invited to contribute to the AQUA (Air QUality Analytics) platform. The AQUA platform is an air pollution data analytics solution. AQUA users are policy-makers and public servants responsible for monitoring the pollution levels in different areas of the country.

In addition to this project specification, you are also provided a code template called `project.zip` (available on ELE). The code template provides the file structure for your submission as well as the datasets to be used in this project. You will notice that the files will have incomplete function declarations. You will write your code within the provided template making the necessary changes to function signatures. Notice that all your pytest code should go within the `test` directory of the template.

3 The datasets

You will be provided two different types of data, namely the *air pollution data* and the *road network data*. The different datasets will be used as described in each of the problems specifications. Below we provide an overview to datasets.

3.1 Air pollution data

The air pollution dataset contains one year (2021) of hourly measures for *three* different pollutants, namely **nitric oxide** (NO) (referred to as `no`), **PM10 inhalable particulate matter** ($\leq 10\mu m$) (referred to as `pm10`) and and **PM2.5 inhalable particulate matter ($\leq 2.5\mu m$) (referred to as `pm25`). You will be provided the data files for three different measuring sites in London, each of them containing ($24 \times 365 = 8760$) data points.

The data files will be provided in a CSV (comma-separated values) format, in which each line corresponds to a data point (e.g., measurement) and the columns are separated by commas. The first row will contain the column names (i.e., `date,time,no,pm10,pm25`). The first two columns (`date` and `time`) correspond to the date and time of the measurement. The `date` is represented in a `YYYY-MM-DD` format (year-month-day) while time is represented as `HH:MM:SS` (hour:minute:second). See below an example of the pollution dataset.

date	time	no	pm10	pm25
2021-01-01	01:00:00	6.08624	21.3	18.6
2021-01-01	02:00:00	7.40564	45.1	40.3
2021-01-01	03:00:00	5.01157	25.6	23.9
2021-01-01	04:00:00	12.76834	20.8	18.1
2021-01-01	05:00:00	9.09745	20.9	21.7

The monitoring sites

- **Marylebone Road**

- Filename: `Pollution-London Marylebone Road.csv`
- Type: Urban Traffic
- Coordinates (longitude, latitude): $(-0.154611, 51.52253)$
- Site code: MY1

- **N. Kensington**
  - Filename: `Pollution-London N Kensington.csv`
  - Type: Urban Background
  - Coordinates (longitude, latitude): $(-0.213492, 51.52105)$
  - Site code: KC1
- **Harlington**
  - Filename: `Pollution-London Harlington.csv`
  - Type: Urban Industrial
  - Coordinates (longitude, latitude): $(-0.441614, 51.48879)$
  - Site code: HRL

3.2 Road network

The road network data is provided in the form of a geospatial image in which the availability of data regarding the pavement/sidewalk is encoded as different colours. The Figure 1 shows the roads within a $1km$ radius from the Marylebone Road station. The red lines in the Figure 1 correspond to the roads with pavement on both sides. The cyan lines, on the other hand, represent the roads where there is no information regarding pavement availability. For your tests, you will be provided the image `map.png` for the Marylebone Road monitoring station.

4 Project specification

You will implement the analytics routines of the AQUA platform which must generate an array of statistics for different air monitoring stations. More precisely, you will have to implement three modules of the platform:

- Pollution Reporting (PR)
- Mobility Intelligence (MI)
- Real-time Monitoring (RM)

The requirements for each of the modules are specified in the following sections.

4.1 The *Pollution Reporting (PR)* module

One of the key functionalities of the AQUA platform is its Pollution Reporting (PR) module whose main responsibility is providing relevant information regarding the pollution levels at the different monitoring stations. From the PR module users can generate an array of statistics at different aggregation levels, namely by *hour, day, month, week and day of the week.* In these analyses you will be using temporal data and therefore you may find useful libraries such as the built-in `datetime` and `calendar` or modules with similar functionalities from `numpy`.

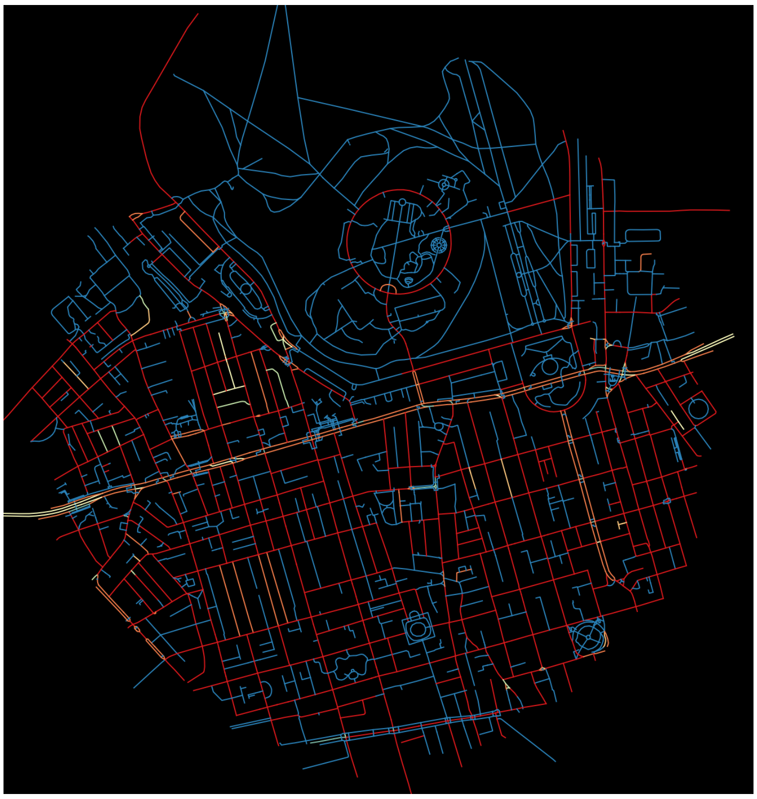

Figure 1: Example of the road network data for the Marylebone Road area

4.1.1 Coding tasks

Module file: `reporting.py`

In the routines below, you **can** use `numpy`.

- Pollution analyses functions.

  `daily_average(data, monitoring_station:str,pollutant:str)`: returns a list/array with the **daily** averages (i.e., 365 values) for a particular pollutant and monitoring station.
  [4 marks]

  `daily_median(data, monitoring_station:str,pollutant:str)`: returns a list/array with the **daily** median[1] values (i.e., 365 values) for a particular pollutant and monitoring station.
  [5 marks]

  `hourly_average(data, monitoring_station:str,pollutant:str)`: returns a list/array with the **hourly** averages (i.e., 24 values) for a particular pollutant and monitoring station.
  [5 marks]

  `monthly_average(data, monitoring_station:str,pollutant:str)`: returns a list/array with the **monthly** averages (i.e., 12 values) for a particular pollutant and monitoring station.
  [4 marks]

  `peak_hour_date(data, date:str, monitoring_station:str,pollutant:str)`: For a given date (e.g., `2021-01-01`) returns the hour of the day with the highest pollution level and

[1] `https://en.wikipedia.org/wiki/Median`

its corresponding value (e.g., `(12:00, 14.8)`).

[4 marks]

- Handling missing data

  `count_missing_data(data, monitoring_station:str,pollutant:str):` For a given monitoring station and pollutant, returns the number of `'No data'` entries are there in the data.

  [4 marks]

  `fill_missing_data(data, new_value, monitoring_station:str,pollutant:str):` For a given monitoring station and pollutant, returns a copy of the data with the missing values `'No data'` replaced by the value in the parameter `new_value`.

  [4 marks]

[Total 30 marks]

4.2 The *Mobility Intelligence (MI)* module

The Mobility Intelligence module is crucial for the larger ambitions of tackling and reducing the GHG emissions related to transport. One key characteristic of greener cities is their *walkability*, i.e., how easy it is for the residents of a city to use walking as their primary mobility mode. The AQUA platform will contribute to this effort through its Mobility Intelligence module. This module is responsible for the analyses of the road infrastructure around one of the three monitoring stations.

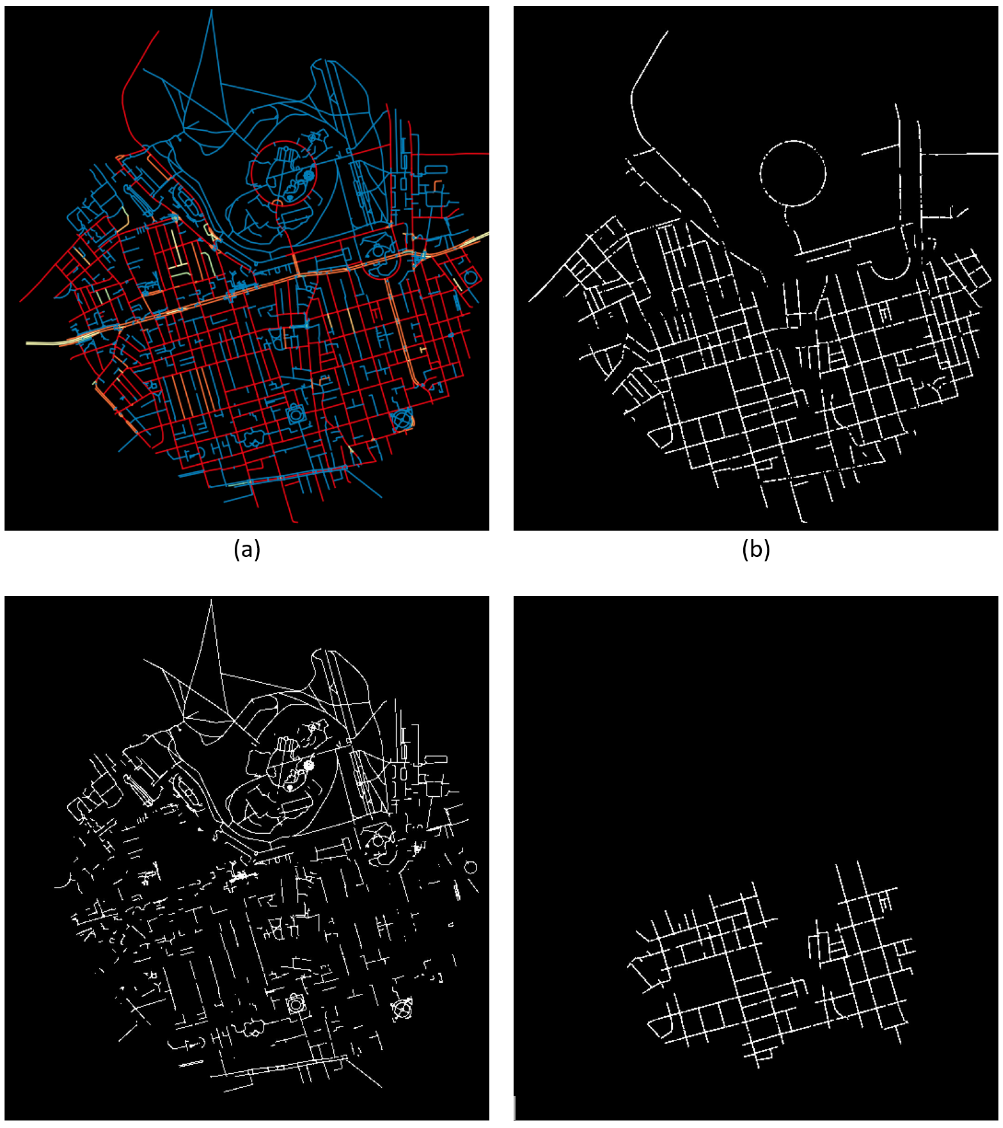

Figure 2: (a) Input image of the Marylebone Road area map. (b) All the red pixels in (a) are shown as white. Other pixels are black. (c) All the cyan pixels in (a) are shown as white. Other pixels are black. (d) The top two largest connected components detected from (b).

Task MI-1: (Related topics: control flow and double for-loop)

Figure 2a shows a two-dimensional image of the Marylebone Road area map containing a two-dimensional pixel (picture element) array. In this image, different pavement types are highlighted in different colours, for example, red colour for pavement in both directions.

Each image pixel contains three values corresponding to the values for red, green, and blue (RGB) colours. Each colour ranges between 0 and 255. For example, a pixel contains the values of 200, 20, and 20 for RGB colours, respectively. This pixel appears red in the image because the red colour value is large and other colour values are small. This task is about using the RGB colour information to identify pixels of interest.

In this task, you must use the double for-loop to traverse all the pixels in the image to find the pixels of interest. Here is the rule for deciding a red pixel. Denote the RGB colour values as r, g, and b, respectively. If, at a pixel, $r > UpperThreshold$ and $g < LowerThreshold$ and $b < LowerThreshold$, mark this pixel as red. If, at a pixel, $r < LowerThreshold$ and $g > UpperThreshold$ and $b > UpperThreshold$, mark this pixel as cyan.

4.2.1 Coding tasks

Module file: `intelligence.py`

- **Task MI-1a**: Your task is to read a city map image (e.g., Figure 2a), specified by the filename, `map_filename` (e.g., `='map.png'`), find all the red pixels from the image, and output a binary image file as map-red-pixels.jpg. The expected output is shown in Figure 2b as an example.

  `find_red_pixels(map_filename, upper_threshold=100, lower_threshold=50)`: returns a 2D array in `numpy` representing the output binary image and writes the 2D array into a file named as **map-red-pixels.jpg**.

  [2 marks]
- **Task MI-1b**: Your task is to read a city map image (e.g., Figure 2a), specified by the filename, `map_filename` (e.g., `='map.png'`), find all the cyan pixels from the image, and output a binary image file as map-cyan-pixels.jpg. The expected output is shown in Figure 2c as an example.

  `find_cyan_pixels(map_filename, upper_threshold=100, lower_threshold=50)`: returns a 2D array in `numpy` representing the output binary image and writes the 2D array into a file named as **map-cyan-pixels.jpg**.

  [2 marks]

[Total 4 marks]

Task MI-2: (Related topics: queue-like ndarray and sorting)

You need the outputs obtained in Task MI-1 to complete the tasks in this section. The tasks are given as follows.

Task MI-2a: Assume that there is a set of pixels S in an image, as shown in Figure 3a. For any two pixels, a and b, in S, if there is a path connecting pixels a and b and the path is entirely contained in S, then S is called a connected component. In this example, as shown in Figure 3a, there are three connected components, S, T and U, in the image with different connected component region sizes.

Your task is to find all the 8-connected components from the output image obtained in the previous task, for example, Figure 2b, with the assumption of 8-adjacency (see Figure 3b). For each connected component region, you need to output the number of pixels inside it, and write the number

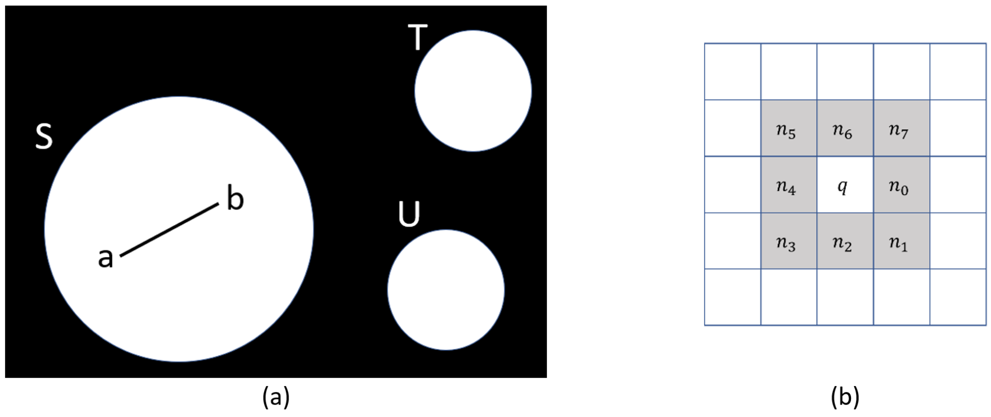

Figure 3: (a) An example with three connected components, S, T and U. (b) The 8-neighbours, n, of q.

of pixels into a text file cc-output-2a.txt. In cc-output-2a.txt, the last line should give the total number of connected components. The expected output is as follows.

```
Connected Component 1, number of pixels = 892
Connected Component 2, number of pixels = 700
Connected Component 3, number of pixels = 208
⋮
Connected Component 223, number of pixels = 152
Total number of connected components = 223
```

The algorithm outline for detecting the connected components is given in **Algorithm 1** (Figure 4).

Algorithm 1. Connected Component Detection (you need to modify it in this task)

Input: A binary image IMG, for example, Figure 2b. In this binary image, white colour (a value of 255) represents the pavement region, and black colour (a value of 0) represents the background region.

Outputs: (1) All 8-connected components, which are detected from IMG and stored in a 2D array *MARK*. (2) The corresponding number of pixels inside each connected component region.

```
1. Set all the elements in $MARK$ as unvisited, i.e., 0.
2. Create an empty queue-liked ndarray $Q$.
3. for each pixel $p(x,y)$ in $IMG$ do #starting from the top left corner of $IMG$, row-by-row scanning
      if $p(x,y)$ is the pavement pixel and $MARK(x,y)$ is unvisited then
         set $MARK(x,y)$ as visited;
         add $p(x,y)$ into $Q$;
         while $Q$ is not empty do
            Remove the first item $q(m,n)$ from $Q$;
            for each 8-neighbour $n(s,t)$ of $q(m,n)$ do
               if $n(s,t)$ is the pavement pixel and $MARK(s,t)$ is unvisited then
                  set $MARK(s,t)$ as visited;
                  add $n(s,t)$ into $Q$;
            end for
         end while
   end for
```

Figure 4: The algorithm outline for detecting the connected components.

Requirements:

- Your code must follow the algorithm outlined in Algorithm 1 (Figure 4).
- Algorithm 1 marks all the connected components. In other words, it assigns all the connected components with the same value, e.g., 1 (or other non-zero value) = *visited*, in the output image. In this task, you should improve and modify Algorithm 1 in order to output all the detected connected components and their corresponding numbers of pixels.
- You must use ndarray in `numpy` to create the queue Q in Algorithm 1.
- You cannot use any pre-defined or third-party connected component functions.
- You need to implement the summation function by yourself. Any built-in functions and third-party functions related to the summation operation such as sum(), numpy.sum() are not allowed to use.

Task MI-2b: Your task is to perform sorting on the connected component region size based on the number of pixels and output all connected components in descending order. For each connected component region, you need to output the number of pixels inside it, and write the number of pixels into a text file cc-output-2b.txt. In cc-output-2b.txt, the last line should give the total number of connected components. The expected output is as follows.

```
Connected Component 130, number of pixels = 12364
Connected Component 110, number of pixels = 8172
Connected Component 117, number of pixels = 4716
⋮
Connected Component 10, number of pixels = 4
Total number of connected components = 223
```

Output the top two largest connected components (Connected Components 130 and 110 above) in the image, for example, Figure 2d.

Requirements: You must implement the sorting function by yourself, and you cannot use any pre-defined, built-in, or third-party sorting functions, e.g., sorted().

4.2.2 Coding tasks

Module file: `intelligence.py`

- **Task MI-2a (cont'd)**:

  `detect_connected_components(IMG):` reads IMG returned from Task MI-1, returns a 2D array in `numpy` MARK and writes the number of pixels inside each connected component region into a text file **cc-output-2a.txt**. **[10 marks]**

  Documentation: To describe the improvements and modifications on the Algorithm 1. **[2 marks]**

- **Task MI-2b (cont'd)**:

  `detect_connected_components_sorted(MARK):` reads MARK returned from Task MI-2a, writes all connected components in decreasing order into a text file **cc-output-2b.txt**, and writes the top two largest connected components into a file named as **cc-top-2.jpg**. **[4 marks]**

[Total 16 marks]

4.3 The *Real-time Monitoring* module

For this module you were requested to display real-time analytics based on data obtained from the London Air API (`https://www.londonair.org.uk/Londonair/API/`). The API provides access to hourly data for different monitoring stations. The RM module will use the `requests` library to retrieve data from the API endpoint. The RM module, however, has one single requirement: provide real-time statistics for different monitoring stations and pollutants. It means that for this part of the assignment you will have the opportunity to conceive, design and implement the functionalities it will have. It can range from displaying a single value refreshing it whenever new data is available, to a text-based, customizable dashboard showing multiple features.

In this part of the assignment, your work will be assessed based on the following criteria:

Breadth (30%): the diversity of programming concepts being demonstrated in your code (e.g., loops, conditionals, data structures, regex, anonymous functions, list comprehension, etc.) across all the functionalities you implemented. **Notice that not every single function you implement is expected to cover all the concepts.**

Readability (20%): how readable is your code. Do you follow a consistent coding style? Do you use in-line comments throughout your code? Are your variable and function names meaningful?

Documentation (20%): how well documented are your functions. Are you using docstrings? Are your docstrings describing the function, its parameters and returns?

Relevance (10%): how relevant and suitable are your functionalities in the context of the AQUA platform and its role as a pollution monitoring platform? Are the functionalities you implemented providing relevant and useful information to the users?

Creativity (10%): how innovative, surprising and creative are you functionalities? How do the data obtained from the API is combined with other pieces of data to provide richer, non-trivial pieces of information?

Presentation (10%): how is your data being presented? Are they formatted in a way that can be easily understood and interpreted by the user, even in a text-based interface?

4.3.1 Coding tasks

Module file: `monitoring.py`

For this module you are expected to implement **at least four** different functions for the RM module. Notice that the names used below are just placeholders. You should name the functions appropriately according to the task they perform.

rm_function_1(*args,kwargs):** **your specification**

rm_function_2(*args,kwargs):** **your specification**

...

rm_function_n(*args,kwargs):** **your specification**

[Total 20 marks]

4.4 The user interface

The ACQUA system should interact with the user through a text-based interface. The user should use the keyboard to navigate through the different options. Upon initialisation, the system should show a menu and prompt the user to choose an option. The main menu should show the options in a format similar to the example below.

- R - Access the PR module
- I - Access the MI module
- M - Access the RM module
- A - Print the *About* text.
- Q - Quit the application

4.4.1 Coding tasks

Module file: `main.py`

Within the module you should implement the following functions:

`main_menu()` A function that will be executed upon the initialisation of the program, showing the main menu of the program allowing the user to navigate through the different options.
[2 marks]

`reporting_menu()` A function that will be executed when the user chooses the ‘R‘ option in the main menu. It should allow the user to choose the necessary options to perform the analyses described in Section 4.1, allowing the user to navigate through the different options and return to the main menu.
[2 marks]

`intelligence_menu()` A function that will be executed when the user chooses the ‘I‘ option in the main menu. It should allow the user to choose the necessary options to perform the tasks described in Section 4.2, allowing the user to navigate through the different options and return to the main menu.
[2 marks]

`monitoring_menu()` A function that will be executed when the user chooses the ‘M‘ option in the main menu. It should allow the user to choose the necessary options to perform the tasks you implemented for the RM module described in Section 4.3, allowing the user to navigate through the different options and return to the main menu.
[2 marks]

`about()` A function that will be executed when the user chooses the ‘A‘ option in the main menu. It should print a string containing the module code (*ECM1400*) and a `string` with your 6-digits candidate number (including any leading zeroes) returning to the main menu afterwards.
[1 marks]

`quit()` A function that will be executed when the user chooses the ‘Q‘ option in the main menu. It should terminate the program.
[1 marks]

[Total 10 marks]

4.5 Utility functions

In addition to the specific functions relative to each of the AQUA modules, it is also a good practice to have a `utils` module containing useful functions that will be used in many different parts of the software.

4.5.1 Coding tasks

Module file: `utils.py`

`sumvalues(values)` A function that receives a list/array and returns the sum of the values in that sequence. The function should **raise an exception** if non-numerical values are present in the list. You have to implement your function from the basics (i.e., without using pre-defined functions)

[2 marks]

`maxvalue(values)` A function that receives a list/array and returns the index of the maximum value in that sequence. The function should **raise an exception** if non-numerical values are present in the list. You have to implement your function from scratch (i.e., without using pre-defined functions)

[2 marks]

`minvalue(values)` A function that receives a list/array and returns the index of the minimum value in that sequence. The function should **raise an exception** if non-numerical values are present in the list. You have to implement your function from scratch (i.e., without using pre-defined functions)

[2 marks]

`meannvalue(values)` A function that receives a list/array and returns the arithmetic mean value in that list/array. The arithmetic mean μ of a list with n elements can be defined as

$$\mu = \frac{1}{n}\sum_{i=0}^{n-1} a[i] = \frac{a[0] + a[1] + \cdots + a[n-1]}{n}$$

The function should **raise an exception** if non-numerical values are present in the list. You have to implement your function from scratch (i.e., without using pre-defined functions)

[2 marks]

`countvalue(values, x)` A function that receives a list/array `values` and a value `x` and returns the number of occurrences of the value `x` in the list/array `values`. The function should return 0 if the value is not present in the list/array. You have to implement your function from scratch (i.e., without using pre-defined functions)

[2 marks]

[Total 10 marks]

5 Coding practices

You are expected to implement your code using good coding practices. Therefore, 10 marks will be awarded based on the quality of your code and your good coding practices. To get full marks, all your functions should include unit testing and be fully documented.

- Testing: As your software will depend on third-party services and is designed to run continuously there is a good chance errors will occur while the program is deployed. As part of your code you should include unit testing using `pytest` for each of your functions. **[5 marks]**
- Documentation: The service you develop may be useful for more than just one user and it may inspire others to extend the code. A user must know how to use the system and a developer must know how the code is structured, what it does and how to extend it. Thus, all functions you implement are expected to contain both in-line comments as well as docstrings.

[5 marks]

[Total 10 marks]