# ECM1400 Programming

python™

# Coursework Q&A

# Implementation 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.

# Submission instructions

- 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.

# Coursework overview

- You will have to design and implement an air pollution analytics platform.
- You were provided
  - Project specification
  - Data files
  - Code template
- The solution will have three main components
  - Pollution Reporting
  - Mobility Intelligence
  - Real-time Monitoring

# The assessment environment

- Your code will be tested in a Python 3.9 environment containing, in addition to Python's standard library, the following additional modules:
  - Numpy
  - Pandas
  - Scikit-image
  - Requests
- You will be taught how to use the basic functionalities of Numpy and Scikit-image.
  - Pandas
    - Won't be covered in lectures but if you know how to use it, you will be allowed to, unless stated otherwise.
  - Requests
    - You were provided a function that can make a query to the API using the Requests library
    - Not part of the standard library. You have to install it.
    - Requires Python 3.7 or above

# Data overview

- Pollution data
  - You will be provided three pollution data files  for three different pollutants from three different pollution monitoring stations
  - One year, hourly readings

| 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 |

# Data overview

- Road network data
  - Area with 1km radius around the Marylebone Rd monitoring station.
  - Pavement availability encoded in different colours

# Pollution reporting module

# Overview

- In this module you are required tasks related to the past pollution levels for the different monitoring stations
  - Perform different types of aggregations and statistics on the data.
  - Handle missing data points

# 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 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]**

# Mobility Intelligence Module

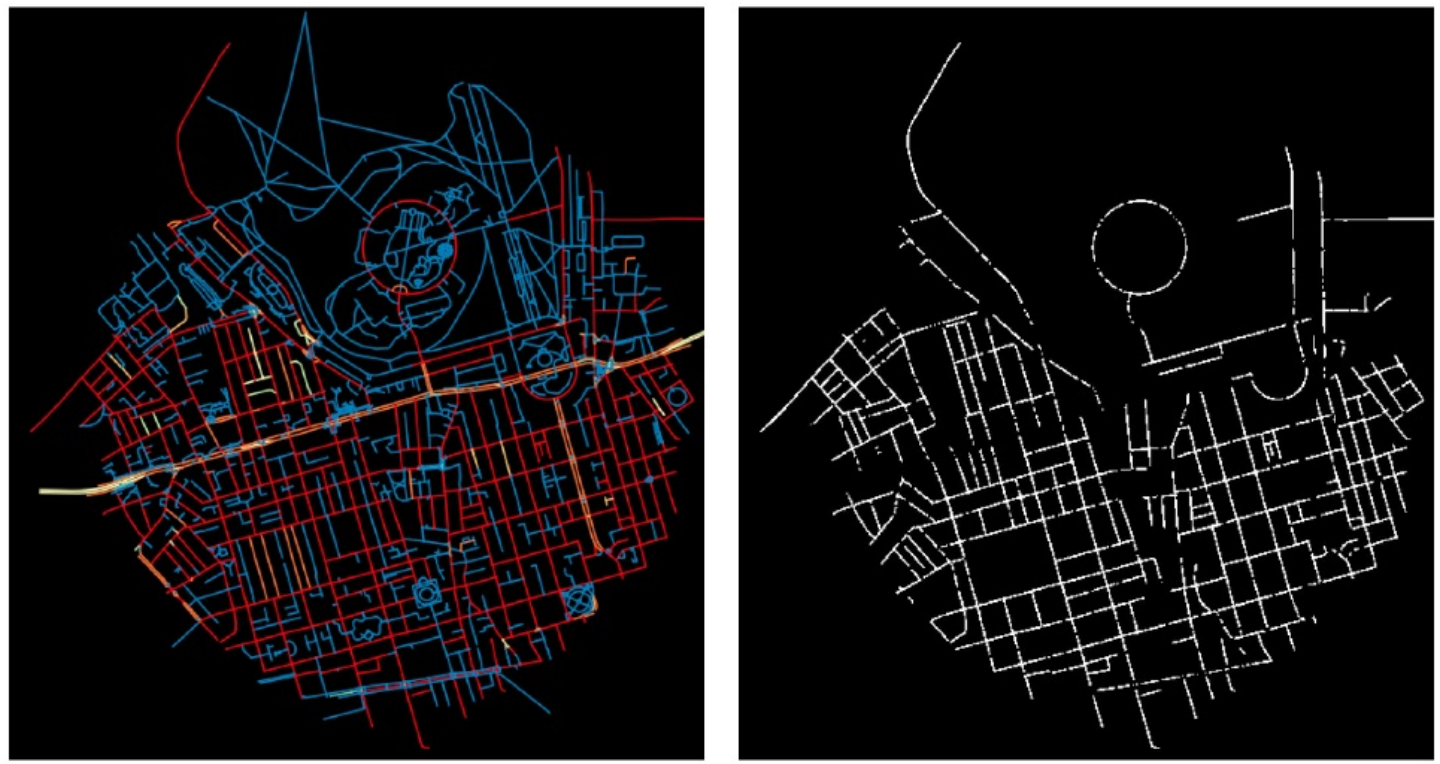
# The Mobility Intelligence (MI) module

- Pavement detection is essential in urban analytics, e.g., walkability.
- In the figure, different pavement types have been highlighted in different colours.

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**.



(a) (b)

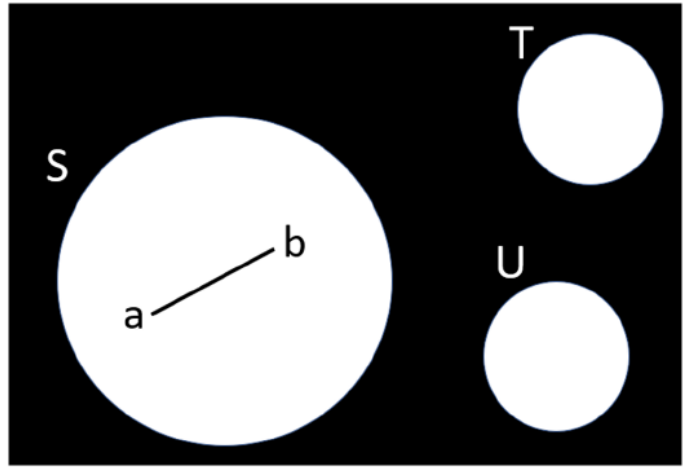
- **Task MI-1b**: Your task is to read a city map image (e.g., Figure 2a), specified by the filename, `map_filename` (e.g., `='map.png'`), find all the cyan pixels from the image, and output a binary image file as map-cyan-pixels.jpg. The expected output is shown in Figure 2c as an example.

  `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**.

(a)

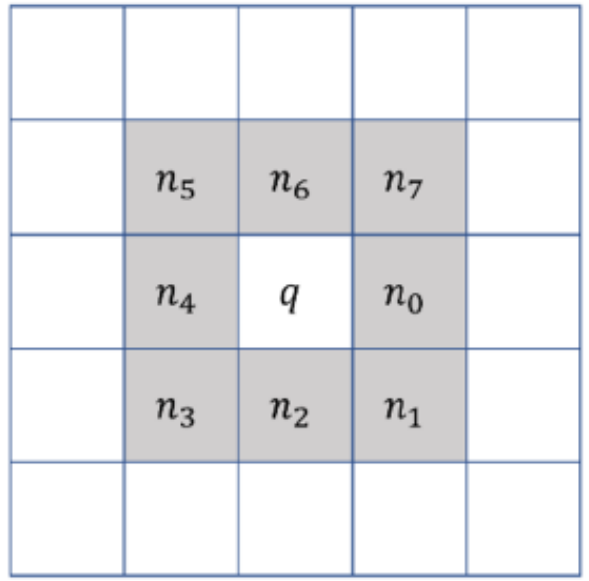
(c)

**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.



(a)

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 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.




- **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.

Connected Component 1, number of pixels = 110
Connected Component 2, number of pixels = 453
Connected Component 3, number of pixels = 101
Total number of connected components = 3

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

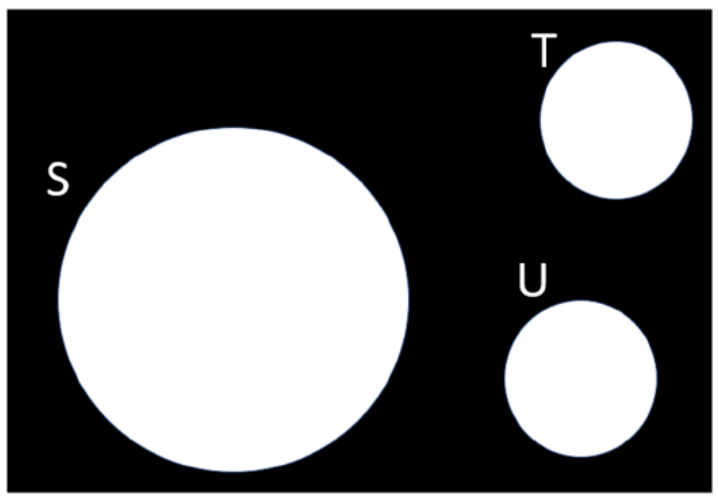
**Input:** A binary image $IMG$, for example, Figure 2b. In this binary image, white colour (a value of 255) represents the pavement region, and black colour (a value of 0) represents the background region.

**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.

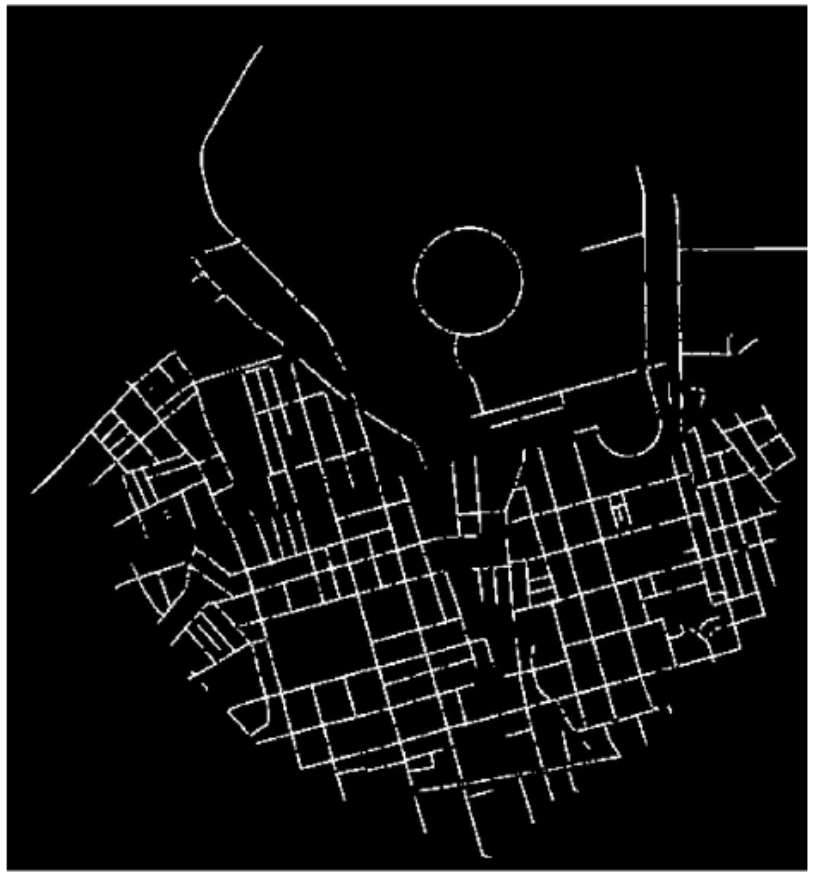
1. Set all the elements in *MARK* as unvisited, i.e., 0.
2. Create an empty queue-liked ndarray $Q$.
3. **for** each pixel $p(x, y)$ in $IMG$ **do** #starting from the top left corner of $IMG$, row-by-row scanning
   **if** $p(x, y)$ is the pavement pixel and $MARK(x, y)$ is unvisited **then**
   set $MARK(x, y)$ 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**

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.



(a)



(b)

```
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
```

**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.

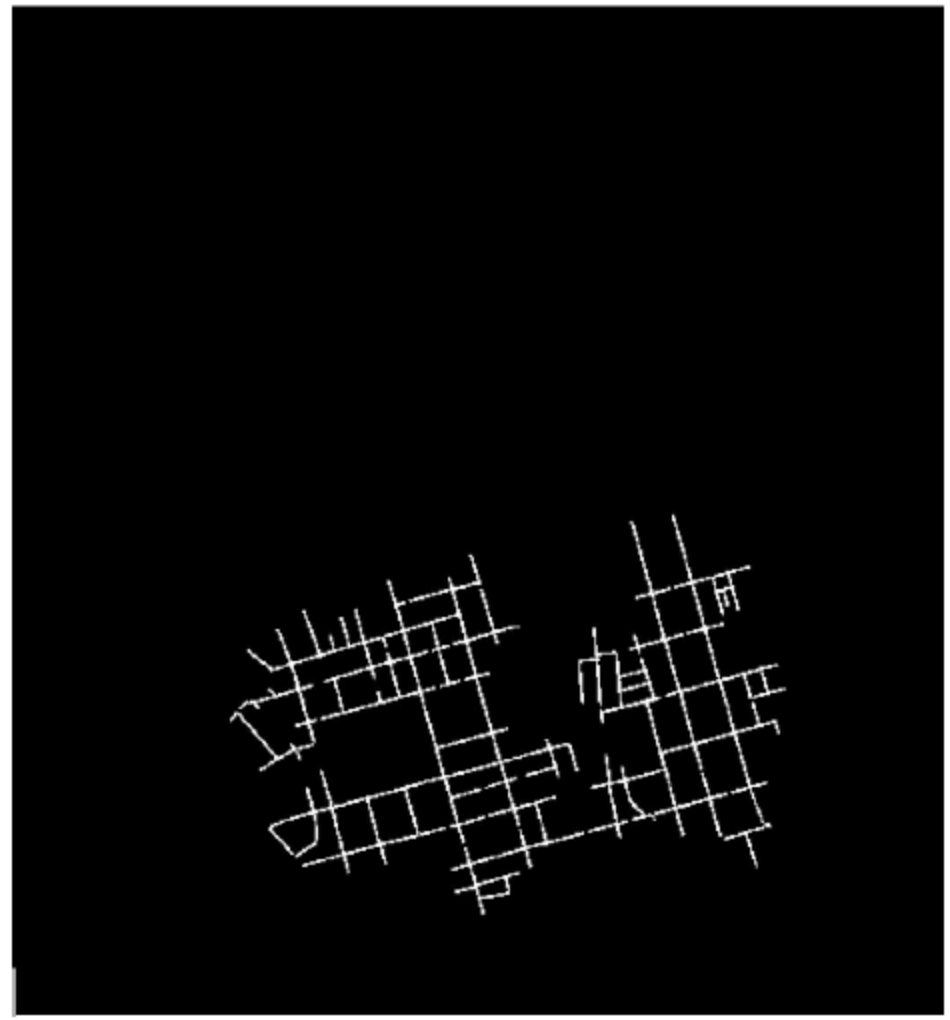
```
Connected Component 130, number of pixels = 12364
Connected Component 110, number of pixels = 8172
Connected Component 117, number of pixels = 4716
⋮
Connected Component 10, number of pixels = 4
Total number of connected components = 223
```

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().

- **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**.



(d)

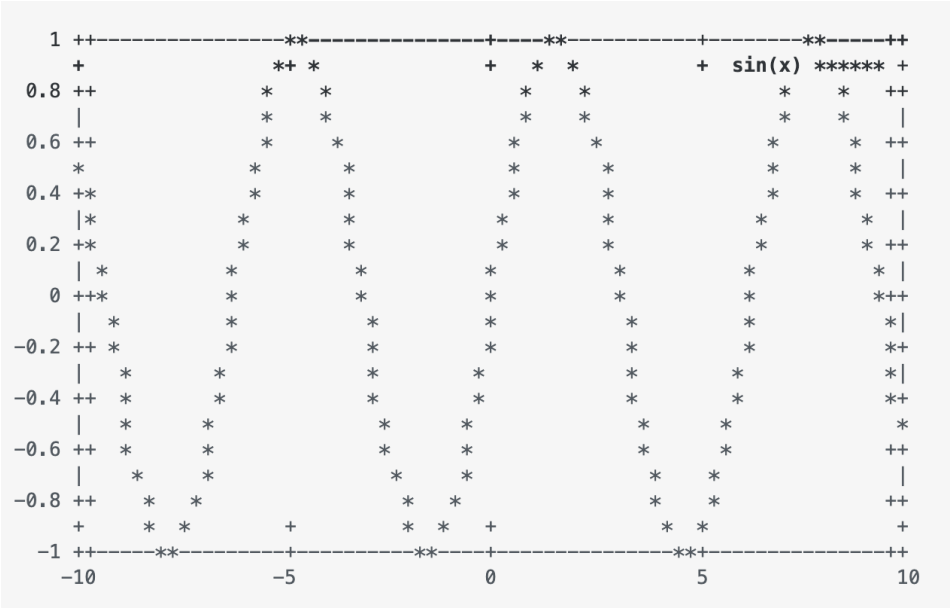
# Real-time Monitoring module

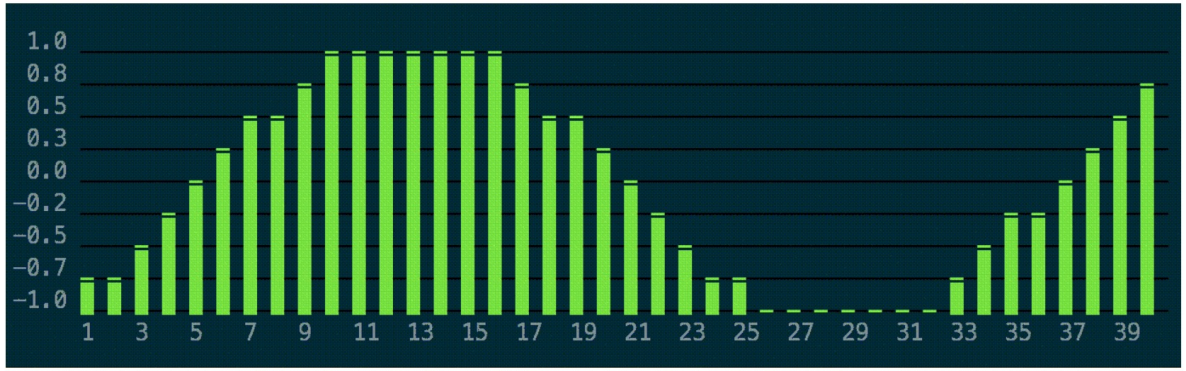
# Overview

- This software module will allow you to demonstrate your creativity and programming skills.
- Your submission will be assessed based on the criteria below:
  - The **breadth** diversity of programming concepts being demonstrated in your code
  - How **readable** is your code.
  - How **well documented** are your functions
  - How **relevant** and suitable are your functionalities in the context of the AQUA platform and its role as a pollution monitoring platform
  - 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?
  - 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?

# Real-time monitoring

- There are many things you can do
- Be creative
- Combine datasets
- Document your code well



# User interface

# User interface

- You will have to implement a text-based interface that will allow users to use the different functionalities of the system.

```
R - Pollution Reporting
I - Mobility Intelligence
M - Real-time Monitoring
A - About
Q - Quit

Choose an option:
```

```
0 - Marylebone Road
1 - N. Kensington
2 - Harlington

Choose a monitoring station:
```

# Utils

# Utils

`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]**

# Utils

`meannvalue(values)` A function that receives a list/array and returns the arithmetic mean value in that list/array. The arithmetic mean $\mu$ 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)

# FAQ

# FAQ

- In addition to the standard library and the other ones you mentioned, can I use this third-party library xyz?
  - No. The assessment was designed in a way for you to demonstrate mastery of the concepts we covered in the lectures.

- Can I change the names or the signatures of the functions in the template?
  - No (Except for the placeholder functions `rm_function` in the `monitoring.py` module)! You must use the functions as they are defined in the template (typos included). We will be using automated testing to assess whether your function is working as expected and if the test fails, marks will be deducted.

- Can I implement additional functions?
  - Yes. You can implement additional functions as long as the ones in the specification are also implemented and return the expected values.

- Can I use classes/decorators/other concepts not covered by the module?
  - Yes. But the required functions will have to be implemented and return the expected results. If the required functionalities are implemented elsewhere, it may impact the readability of your code, which can affect your mark.

# FAQ

- What additional libraries do I **need** to install to complete this assignment?
  - Scikit-image (to read/write images [to be covered])
  - Numpy (to represent the images as arrays[to be covered])
  - Requests (to query the monitoring API [code already provided])
  - Pytest (to write your tests [to be covered])
- What other libraries **can I** install?
  - Pandas [won't be covered]