CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

BIG DATA & ANALYTICS

ASSIGNMENT 2*:
MAP-REDUCE, SPARK CORE AND SPARK STREAMING

***This assignment has been modified after the approved Department Assessment Plan**

MARK BREAKDOWN.

After the Department Assessment Plan the assignment is now worth 50 marks:

- Part1, Part2, Part3 and Part4 => 25 marks
  - Part1 – Exercise 1 => 6.25 marks.
  - Part2 – Exercise 2 => 6.25 marks.
  - Part3 – Exercise 3 => 6.25 marks.
  - Part4 – Exercise 4 => 6.25 marks.

- **Part5 => 25 marks**
  - **Part5 – Exercise 5 => 10 marks.**
  - **Part5 – Exercise 6 => 10 marks.**
  - **Part5 – Exercise 7 => 5 marks.**

SUBMISSION DETAILS.

Submission deadline: Sunday 10th of May, 11:59pm
Please submit to Canvas (folder 5. Submissions) a zip file **A02.zip** containing the following Python files:

- Part 1 -> A02_Part1.py
- Part 2 -> A02_Part2.py
- Part 3 -> A02_Part3.py
- Part 4 -> A02_Part4.py
- **Part 5 -> A02_Part5.py**

MY_CODE FOLDER

The assignment is divided into **5 parts**:
- The original 4 parts.
- 1 extra part due to the approved Department Assessment Plan.

- Part1: Introductory Data Analysis with Spark Core.
- Part2: Introductory Data Analysis with Spark Core.
- Part3: Advance Data Analysis with Spark Core.
- Part4: Introductory Data Analysis with Spark Streaming.
- **Part5: Map-Reduce, Spark Core and Spark Streaming.**

The five parts are provided in the folder “my_code”.
Each part provides a Python file (A02_Part_Number.py) with the exercises to be completed.

MY_RESULT FOLDER

For Part1, Part2, Part3 and Part4 you can find the correct results to be obtained in the folder “my_result”.

ASSIGNMENT 2 – PART 1, PART 2, PART 3 and PART4

COCA-COLA BIKES DATASET.

Cork Smart Gateway is the home of open data for Cork: http://data.corkcity.ie/
At the moment it contains 8 datasets in open format. One of them is the "Coca-Cola Zero Bikes" (http://data.corkcity.ie/dataset/coca-cola-zero-bikes) which provides data related to the bike-sharing rental service supported by the city council.

My former colleague Michael O'Keefe collected data from mid January 2017 to late September 2017 by creating a service quering the API every 5 minutes from 6am to midnight and gathering all entries of a day into a file.

I have selected the files for the period 01/02/2017 – 31/08/2017 and provided the entries in the following csv format:
status ; name ; longitude ; latitude ; dateStatus ; bikesAvailable ; docksAvailable
For example, the aforementioned entry for Kent Station would be represented as follows:
0;Kent Station;-8.45821512;51.90196195;15-10-2018 15:32:30;3;23

In total, the dataset for the selected period contains 1,339,200 entries for 43,200 API requests over 200 days.

ASSIGNMENT 2 – PART 1 **(Week 9)**

GOAL.

Given the Coca-Cola bikes dataset provided:

- Create a Spark job to compute the number of times each Coca-Cola bike station ran out of bikes. Sort the stations by decreasing number of ran outs.
    - Note: A bike station is ran out of bikes if:

        status == 0 and *bikes_available == 0*

EXERCISE 1.

To perform the Spark job, complete the following code:

- **A02_Part1.py =>** Program the function **my_main**.

Note: You can use as many auxiliary functions as you need.

ASSIGNMENT 2 – PART 2 **(Week 10)**

GOAL.

Given the Coca-Cola bikes dataset provided:

- Create a Spark job to compute the amount of times per day and hour that the Fitzgerald Park station was run out of bikes. Present this result as the total amount and as the percentage of total ran outs the station had across the dataset.
    - Note: A bike station is ran out of bikes if:

        status == 0 and *bikes_available == 0*

EXERCISE 2.

To perform the Spark job, complete the following code:

A02_Part2.py => Program the function **my_main**.

Note: You can use as many auxiliary functions as you need.

ASSIGNMENT 2 – PART 3 **(Week 11)**

GOAL.

Given the Coca-Cola bikes dataset provided:

- Create a Spark job to compute the actual ran-out times for the Fitzgerald Park station. Sort the ran-out times by increasing time in the calendar.
    - We define an "actual" ran-out time as the first moment in which our station is measured to be out of bikes. We define further measurements where the station still has no bikes to be "continuations" of the actual ran-out time previously measured.

      For example: given our 5 minutes measurements, if we notify Fitzgerald Park to be ran-out of bikes at the following times:
        - 10:06:00
        - 10:11:00
        - 15:31:00
        - 15:36:00
        - 15:41:00
        - 19:56:00
    - Then the "actual" ran-out-times are highlighed in red, with "continuations" for them highlighted in blue. In this example, 15:31:00 represents a measurement in which we realised that Fitzgerald Park was ran out of bikes. We can ensure this as 15:26:00 is not in the list. Thus, some Coca-Cola user(s) must took the last bike(s) available at the station between 15:26:00 – 15:31:00.
    - At 15:36:00 we notify that Fitzgerald park has actually ran-out of bikes, and we report it.
    - At 15:41:00 and 15:46:00 we just confirm that the bike station is still with no bikes.
    - Finally, provided that 15:46:00 is not in the list, we can ensure a Coca-Cola user(s) have brought bike(s) back to the station between 15:41:00 – 15:46:00.

      All in all, the goal of the assignment is to print by the screen the actual ran-out times, together with the length (amount of measurements) registered for each of them. In the example before we should have printed:
      - Date (10:06:00, 2)
      - Date (15:31:00, 3)
      - Date (19:56:00, 1)

EXERCISE 3.

To perform the Spark job, complete the following code:

A02_Part3.py => Program the function **my_main**.

Note: You can use as many auxiliary functions as you need.

ASSIGNMENT 2 – PART 4 **(Week 12)**

GOAL.

Given the Coca-Cola bikes dataset provided:

- Adjust the Spark job of Part 1 so that it operates in Streaming mode. In particular:
  - We simulate the dataset to arrive in batches, with each batch consisting of the data of one single day.
  - We reduce our dataset to the processing of just 6 batches, corresponding to the days of the first week of May (from May 1st to May 7th , excluding May 6th for which we unfortunately have no data). This reduction of the dataset is provided automatically by the program via the inclusion of the variable `valid_files`, which is hardcoded to such these days (for the function `streaming_simulation` to operate just on them).

  We want to produce results using the 2 stateful operations seen in class:

- `updateStateByKey`: We want to cummulative display the total amount of ran outs measured during the week. That is, after processing the batch for May 1st we will display just the ran outs for such this day. Then, after processing the batch for May 2nd we will display the cummulative ran outs found in May 1st + May 2nd. After processing May 3rd we will display the the cummulative ran outs found in May 1st + May 2nd + May 3rd (and so on).
- `window`: We want to amalgamate the batches in windows. In particular, we are interested in windows_duration = 2 and sliding_duration = 1. That is, in computing the ran outs for:
  - May 1st - May 2nd.
  - May 2nd – May 3rd.
  - May 3rd – May 4th.
  - May 4th – May 5th.
  - May 5th – May 7th.

  In both cases, we want the results sorted in decreasing order.

EXERCISE 4.

To perform the Spark job, complete the following code:

A02_Part4.py => Program the function **my_model**.

Note: You can use as many auxiliary functions as you need.

ASSIGNMENT 2 – PART 5 (Week 13)

WIKIMEDIA DATASET.

A non-provided dataset contains information on Wikimedia page view statistics for its different projects (e.g., Wikipedia, Wikibooks, Wikivoyage, etc). The dataset contains 100 text files, with each file containing 10,000 lines. Each line of a file contains info for 1 web-page, and it has the following fields and format:

Wikimedia_Project ; Web-page_Name ; Language ; Num_Views

Below is an example of some of the lines of a file:

Wikipedia ; Roger Federer ; French ; 23

WikiBooks ; Sapiens ; English ; 8

Wikipedia ; Taoiseach ; English ; 10

Wikipedia ; Musee du Louvre ; French ; 15

WikiVoyage ; Hawai ; English ; 5

WikiBooks ; Pinocchio ; Italian ; 7

…

Please note that each web-page entry is unique in the entire dataset (i.e., it cannot appear in more than one line of the dataset).

ASSIGNMENT 2 – PART 5 (Week 13)

GOAL.

Program a MapReduce job computing the web-page with most views per project and language. For example, a possible outcome can be:

Wikipedia_English	(Olympic Games, 211)
Wikipedia_French	(Roland Garros, 119)
WikiBooks_Italian	(Pinocchio, 7)

...

EXERCISE 5.

To perform the Map-Reduce job, complete the following code:

- **A02_Part5.py =>** Program the function **my_map**.

> *Note: Assume you have an auxiliary function process_line which, given a line, returns the tuple (project, web-page, language, views) with the info of the line. You do not have to implement process_line, just implement my_map.*

- **A02_Part5.py =>** Program the function **my_reduce.**

Note: Assume you have an auxiliary function get_key_value which, given a line, returns the tuple (key, value) from it. You do not have to implement get_key_value, just implement my_reduce.

ASSIGNMENT 2 – PART 5 (Week 13)

GOAL.

Program a Spark Core job computing the web-page with most views per project and language. For example, a possible outcome can be:

Wikipedia_English	(Olympic Games, 211)
Wikipedia_French	(Roland Garros, 119)
WikiBooks_Italian	(Pinocchio, 7)

...

EXERCISE 6.

To perform the Spark Core job, complete the following code:

- A02_Part5.py => Program the function **my_spark_core_model**.

Note: Assume that sc is the Spark context and that you have an auxiliary function process_line which, given a line, returns the tuple (project, web-page, language, views) with the info of the line. You do not have to implement process_line, just implement my_spark_core_model.

ASSIGNMENT 2 – PART 5 (Week 13)

GOAL.

Assume the dataset arrives in streaming mode: 1 file per minute during a 100 minutes job. We process each file independently (in stateless mode) so as to compute its results.

Program a Spark Streaming job computing the web-page with most views per project and language. For example, a possible outcome can be:

```
-------------------------------
Time Step 1
-------------------------------
Wikipedia_English       (Rafael Nadal, 20)
Wikipedia_French        (Musee du Louvre, 15)
WikiBooks_Italian       (Codice Da Vinci, 4)
...
-------------------------------
Time Step 2
-------------------------------
Wikipedia_French        (Roland Garros, 119)
WikiBooks_Italian       (Pinocchio, 7)
WikiVoyage_English              (Hawai, 9)
...
```

EXERCISE 7.

To perform the Spark Streaming job, complete the following code:

A02_Part5.py => Program the function **my_spark_streaming_model**.

Note: Assume that ssc is the Spark streaming context and monitoring_dir the directory where the new file arrives to. Also assume that you have an auxiliary function process_line which, given a line, returns the tuple (project, web-page, language, views) with the info of the line. You do not have to implement process_line, just implement my_spark_streaming_model.