#1 Session 2.2 : Unit testing with python

General guidelines:

- Most of the proposed exercises will led to the writing of python2.7 functions to be stored in the single script with name *test_S2.py*. *Only when specified, write the programs in other specific scripts.*
- All the provided functions and scripts must be documented using the Doxygen syntax.
- Special note on materials to be submitted for continuous evaluation :
  - All the instructions on script and function names must be rigorously followed. As for continuous integration Automatic code testing will be considered to evaluate your submissions and will rely on the imposed prototypes. Any mistake will then impact on your evaluation.
  - All the codes must have been verified using SonarCloud.
  - Each student must fork the following GitHub project on its own GitHub account `<https://github.com/albenoit/USMB-BachelorDIM-Lectures-Algorithms>`
  - **Once an exercise is done**, the scripts must be pushed to your GitHub repository in folder assignments/Session1.
  - **At the end of the session**, the scripts must be submitted to the main GitHub project using a pull request.

Important note : be warned by all the codes you push on GitHub are open source and any person see it, comment and report on it. Then, copy and past between students and other inappropriate submissions as well as well written algorithms and good ideas will remain accessible to any observer. As a consequence, pay attention to your submissions.

Continuous integration setup:

In order to automate unit testing and code checking, first activate Travis-CI continuous integration on your own GitHub repository. Following guidelines provided on the course slides, configure Travis-CI to launch:

- unit testing and coverage measure using pytest.
- code quality evaluation using SonarCloud (freely available only on public GitHub repositories)
- code coverage reporting using Coveralls (freely available only on public GitHub repositories)

Unit tests:

Get back to the first session assignments. For each exercise, you wrote a function and should have written some basic tests without the use of unit test tools. Then, for each of the assignments:

- Identify all the possible ways to run the function you wrote. In your basic tests, you may not have covered all the use cases.
- Draw a table that lists all the possible use cases and write this in your test script in a dedicated section. The table should have a column that provides a short but meaningful name for each test. *Note that you should ensure that all the computation error cases should be managed AND that any input data specification error should raise an Exception.*
- Implement all those test cases as unit tests using the pytest library and python2.7 in the same script. WARNING, each test function should be named test_originalFunctionName_usecaseName.
- Be sure your function is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.
- Check TravisCI, SonarCloud and Coveralls reports and iteratively improve your tested functions to obtain a perfect reporting.
- To get more insights on your way to write codes, report in the test table how many iterations where required to get this perfect reporting... this number *should* decrease all along this session.