#1 Session 3 : CPU demanding processing

General guidelines:

- Most of the proposed exercises will led to the writing of python3.x functions to be stored in two scripts:
  - the script that implements the processing with name *S3_imgproc_tools.py*.
  - the test script that applies unit tests on all the functions written in the previous script.

  .
- the numpy and OpenCV libraries will be used. If you will to install on your machine, depending on your distribution, you can install them using pip in super user mode : *pip install numpy cv2*. OpenCV version should be above v3.0 to get a nicer python interface. If OpenCV doesn't get installed this way, get a precompiled version to be installed on your OS OpenCV releases and follow the install guide OpenCV python install guide.
- 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-2020>`
  - **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.

Image processing basics : inverse image colors

The aim is to compare three methods able to inverse colors in an image in terms of parameters validation, code complexity and processing speed :

- consider the code snippet proposed in Algorithm 1 to load any image using the OpenCV library :

Algorithm 1 Load an image using the OpenCV library

```
import cv2                                    ▷ import the OpenCV library
import numpy as np                            ▷ import the numpy library
img_gray=cv2.imread('myimage.jpg',0)          ▷ load an image in gray levels
img_bgr=cv2.imread('myimage.jpg',1)           ▷ load an image in Blue Green Red
#display the matrix shapes
print("Gray levels image shape = "+str(img_gray.shape))
print("BGR image shape = "+str(img_bgr.shape))
#display the loaded images
cv2.imshow("Gray levels image", img_gray)
cv2.imshow("BGR image", img_bgr)
cv2.waitKey()
```

- design a function *invert_colors_manual(input_img : ndarray) : returns ndarray*. This function will only rely on raw python with loops without the use of any external library. It takes as input a loaded image and returns the transformed image.
- design a similar function called *invert_colors_numpy(input_img : ndarray)returns ndarray*. This function will rely on the Numpy operators to do the job.
- design a similar function called *invert_colors_opencv(input_img : ndarray)returns ndarray*. This function will rely on the OpenCV operators to do the job. Please have a look at the OpenCV 3.2 Documentation.
- Compare processing speeds when testing the three methods on different image sizes from small to very large (4k images) using the time or timeit modules.
- implement unit tests for each of the three methods. Do those functions need the same tests ? What is the impact on maintainability ?
- Be sure your function is documented following the python standards.
- Commit the code to your local repository and push to your GitHub repository.

Image processing basics : image thresholding

In the same way as with the previous image processing function, one will now consider image thresholding to get a simple binary image.

- design function $threshold\_image\_manual(input\_img : ndarray) : returns\ ndarray$. This function will only rely on raw python with loop without the use of any external library. It takes as input a loaded image and returns the transformed image.
- design a similar function called $threshold\_image\_numpy(input\_img : ndarray) returns\ ndarray$. This function will rely on the Numpy operators to do the job.
- design a similar function called $threshold\_colors\_opencv(input\_img : ndarray) returns\ ndarray$. This function will rely on the OpenCV operators to do the job. More specifically use the OTSU method
- Compare processing speeds when testing the three methods on different image sizes from small to very large (4k images) using the time or timeit modules.
- implement unit tests for each of the three methods. Do those functions need the same tests ? What is the impact on maintainability ?
- Be sure your function is documented following the python standards.
- Commit the code to your local repository and push to your GitHub repository.