#1 Session 0 : Guidelines and source versionning for all the course

General guidelines:

- All the codes and assignments will be related to the following github repository :
`https://github.com/albenoit/BachelorDIM-Lectures-Algorithms-2020`.
- Most of the proposed exercises will led to the writing of python3.x functions to be stored in the single script with name $S1\_algotools.py$. *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.

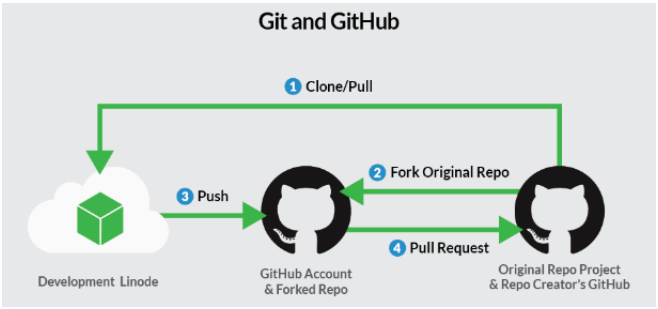

Figure 1: Contributing to a GitHub repository, get your local working copy and your own repository fork. Push local changes to your fork and propose contributions to the original project from you fork with pull requests.

Code versioning:

As for numerous collaborative projects, you will rely on a base repository (here proposed by the teacher), get updates from it and propose contributions from you own copies all along the assignments. As illustrated in the figure above, you will rely on 2 copies, the one on your working computer where you will develop new stuff and another one on your own GitHub account, a *fork* of the original repository that will be fed by your computer copy enhancements (you will *push* updates) and that will propose your contributions to the original repository as a pull request.

This will then secure you code, keep updates from the original repository and feed it with your proposals.

For each of the exercises, you will commit a working version to your fork. After your first push of the first assignment, do a pull request to the course manager repository. **WARNING, be sure that the pull request is entitled like this *YOURFIRSTNAME_ YOURNAME...* with your own firstname and name**. The following code pushes on your fork will then automatically complete your pull first request with extended codes, no worry about any new pull request !

Here are some guidelines:

- on your computer, go to https://github.com and login to your account. Then do the following:
  - Go to the base repository location: `https://github.com/albenoit/BachelorDIM-Lectures-Algorithms-2020`.
  - Clone this repository on your own computer using the Green "Clone or Download" button, copy the provided url into a git client (TurtoiseGit for example). Once entered all the login and passwords, you will have a local copy of the repository.
  - Fork the repository on your own GitHub account using the "Fork" button to get your own copy also on GitHub.

- On your local copy, add a remote repository, the one of your own fork.
  - for Linux guys, use command line from the local project folder and type : *git remote add myfork https://github.com/myaccount/BachelorDIM-Lectures-Algorithms-2020.git*
  - for Windows guys using TurtoiseGit, RightClick on the folder->TurtoiseGit->Settings. Then in the menus, reach Git->Remote and add remote 'myfork'. Accept all following questions...
- Test your repository:
  - make a change in the Readme.md file
  - commit your change locally (Linux : git commit Readme.md -m"this is a test" ; Windows : in the file explorer, right click on the modified file, Git commit->master). Adding a message is mandatory to keep trace of what was done for each change.
  - push your change to your fork on GitHub: Linux : git push myfork master ; Windows, right click on the folder repository, -> Turtoise Git-> Push-> destination, select your fork and validate.
  - Check the result on your GitHub interface, the last commit of the Readme.md file must appear with the message you gave.

  , save the file and try to commit and push to your fork.

You are now ready ! Now keep in mind that :

- If you need to add a new file to the repository, first add it to git before any new commit. Linux guys : git add myfile. Windows guys : right click on the file->TortoiseGit->Add...
- **Once an exercise is done**, the scripts must be pushed to your GitHub repository in the appropriate folder (prefer nice code organization).
- **At the end of the session**, the scripts must be submitted to the main GitHub project using a pull request from your GitHub fork.
- **Important note : be warned that all the codes you push on GitHub are open source and any person can 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.**

#1 Session 1 : Algorithm warm up and introduction to python

Here is a set of basic programs to implement in python. Train and think about efficient programming. Interract with the teacher to get more insights !

Averaging:

One will to compute the average of the *positive elements* of a table.
To this end, Algorithm 1 is proposed.

Algorithm 1 Selective average

```
1: Som ← 0
2: N ← 0
3: for i ← 1 to NMAX do                    ▷ NMAX is the table size
4:     if Tab[i] > 0 then
5:         Som ← Som + Tab[i]
6:         N ← N + 1
7: Moy ← Som/N
8: Display(Moy)
```

- What happens if *Som* initialization is forgotten ?
- What can you expect if all the values are below zero ?
- Translate this algorithm in the python3.x language within script of name *S1_algotools.py*. *You must implement this algorithm as a function that follows prototype "function average_ above_ zero(table:list) returns float".*
- Be sure your function is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.

Table maximum value:

- Propose changes on the previous algorithm to get the maximum value of a table.
- Improve the previous changes by capturing the index of the last maximum value.
- Translate this algorithm in the python3.x language in the same script. *You must implement this algorithm as a function that follows prototype "function max_ value(table:list) returns float, int".*
- Be sure your function is documented following the Doxygen syntax.

- Commit the code to your local repository and push to your GitHub repository.

Reverse a table:

- Define one algorithm able to reverse a table **without the use of any other table**.
- Translate this algorithm in the python3.x language in the same script. *You must implement this algorithm as a function that follows prototype "function reverse_table(table:list) returns table".*
- Be sure your function is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.

Bounding box:

Suppose you receive a binary image (2D matrix with binary values) with true values representing an object and a dark zero valued background.

- Define one algorithm able to compute the bounding box coordinates of the object.
- Translate this algorithm in the python3.x language in the same script. *You must implement this algorithm as a function that follows prototype "function roi_bbox(input_image : numpy array) returns a numpy array of shape 4x2 filled with the four 2D coordinates".*
- Be sure your function is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.

Random array filling :

Suppose you receive a 2D array of shape $N * N$ whose cell type is *char* and that you have a random numbers generator function $alea(v)$

- Propose an algorithm able to fill a specified number of K cells with $'X'$ values at random positions while the others cells should remain empty.
- Translate this algorithm in the python3.x language in the same script. *You must implement this algorithm as a function that follows prototype : "function random_fill_sparse(table:numpy array, K:int) returns numpy array".*
- Be sure your function is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.

Remove whitespace characters in a string:

- Define one algorithm able to parse a string and remove all its whitespace **without the use of any other table**.
- Translate this algorithm in the python3.x language in the same script. *You must implement this algorithm as a function that follows prototype "function remove_ whitespace(table:string) returns string".*
- Be sure your function is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.

Random item selection :

Suppose you receive a list and need to randomly select each of the values **but only once**.

- Propose an algorithm able to **efficiently** randomly select items of a list. Each item must be selected only once and "efficiently" refers to limited memory footprint and low power consumption of the algorithm.
- Translate this algorithm in the python3.x language in the same script. *You must implement this algorithm as a function that follows prototype "function shuffle(list_ in:list) returns list".*
- Be sure your function is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.

A dice game :

A user fights against the computer in a dice game, here are the rules :

- At the beginning, each player has a null score and the winner is the first obtaining at least 100 points.
- Participants play in turn but the user always starts.
- Each player throws a dice as many times as he wants until i) he obtains '1' ii) he wants to stop.
- When stopping, if the player obtained '1' then score does not increase. In the other case, the score is increased by the sum of all the dice values obtained this turn.

Your turn now :

- Propose an efficient algorithm that satisfies those rules.
- Translate this algorithm in the python3.x language in a new script called *S1_ dice.py. You must implement this algorithm as a function that follows prototype "function shuffle(list_ in:list) returns list".*
- Be sure your code is documented following the Doxygen syntax.
- Commit the code to your local repository and push to your GitHub repository.

Sorting :

Many sorting algorithms have been proposed and you use them everyday when exploring file folders, emails, etc. that you need to parse efficiently, sorting by creation date, last change date, etc. We propose to explore two strategies:

1. Selective sort. This algorithm is one of the simplest and consist in finding largest or smallest values and doing permutations. Lets consider a vector that you can divide in two parts, the sorted section that is initially empty and the unsorted part initially filled with the initial unsorted values. Iteratively, the unsorted section is scanned, looking for an extrema. Once found, it is permuted with the first unsorted element and is integrated into the sorted section. Then, iteratively the sorted section grows while the unsorted one is reduced. This method is applied as many times as necessary until the unsorted section is empty.
   (a) Illustrate the algorithm on the following vector sample : 10, 15, 7, 1, 3, 3, 9
   (b) Does the number of iterations depend on the vector content ?
   (c) How many iterations are required to sort the whole vector ?
   (d) How many permutations are applied ?
   (e) How many comparisons are applied ?
   (f) Can you quantify the algorithm complexity ?
   (g) Compare the number of permutations and comparisons for input vectors of varying sizes : 50, 100 and 500
2. Bubble sort. It consists in pair comparisons of all the N elements to be sorted. One permute element at index j with element at index $j+1$ if higher valued. Then when the $N-1$ index is processed, the highest value has migrated to the end of the list. Applying this principle along many iterations, the "lightest" elements gradually migrate to an end of the list and thus inspired the name of the algorithm.
   (a) Illustrate the algorithm on the following vector sample : 10, 15, 7, 1, 3, 3, 9

(b) Does those number of iterations depend on the vector content ?

(c) How many iterations are required to sort the whole vector ?

(d) How many permutations are applied ?

(e) How many comparisons are applied ?

(f) Can you quantify the algorithm complexity ?

(g) compare the number of permutations and comparisons for input vectors of varying sizes : 50, 100 and 500

Hands on sorting !

- Propose an algorithm for each of those sorting methods.
- Translate this algorithm in the python3.x language in a the *S1_algotools.py* script. *You must implement these algorithms as functions that follows prototypes :*
  function sort_selective(list_in:list) returns list
  function sort_bubble(list_in:list) returns list.
- Be sure your function is documented following the Doxygen syntax.
- Within the main function comments, insert all the above questions and their answers.
- Commit the code to your local repository and push to your GitHub repository.

Using SonarCLoud

Here are some more hints to make use of SonarCloud to check your codes. This is complementary to the course slides :

- Get to `https://sonarcloud.io`
- open a free account by login in using your GitHub account
- create a project manually, set a unique project name say "DIMbachelor_Algo_mylogin", the "project Key" should be the repository name (USMB-BachelorDIM-Lectures-Algorithms-2019). and get up to the point where you can download the sonar cloud utility. Copy the example command line in a file, it contains your login for this project... however, do not commit any file with this private login to your github account (public visibility)
- next create file sonar-project.properties filled with (fix the login, organization and project key that are private). You can take inspiration from the one provided in the repository and the example below :

```
# must be unique in a given SonarQube instance
sonar.projectKey=USMB-BachelorDIM-Lectures-Algorithms-2019
# this is the name and version displayed in the SonarQube UI. Was
mandatory prior to SonarQube 6.1.
sonar.projectKey=USMB-BachelorDIM-Lectures-Algorithms-2019
sonar.organization=yourgithubusername-github
sonar.projectName=DIMbachelor_Algo_mylogin
sonar.projectVersion=1.0
# Path is relative to the sonar-project.properties file. Replace "by "/"
on Windows.
# This property is optional if sonar.modules is set.
sonar.sources=.
# Encoding of the source code. Default is default system encoding
#sonar.sourceEncoding=UTF-8
sonar.host.url=https://sonarcloud.io
sonar.login=yourKey, a long string with numbers and letters...
```

- check your email, GitHub should have sent you an alert telling you that a third party app wants to access your repo, check, this should be fine.
- check your sonar cloud web page and check your codes

Once installed, each time you need to check your code, you will do from command line:

- move to your codes directory (cd C:.....)
- lanch sonarcube using command $C : \backslash Users \backslash alben \backslash Desktop \backslash USMB - BachelorDIM - Lectures - Algorithms - 2019 > C : \backslash Users \backslash alben \backslash Desktop \backslash sonar - scanner - 3.2.0.1227 - windows \backslash bin \backslash sonar - scanner.bat$
- get to the sonarcloud project web page to check code quality.