#1 Session 0

First evaluate your initial level with the Moodle on-line evaluation proposed at `<https://moodle2.iut-acy.univ-savoie.fr>`.
Follow this procedure:

1. login into the platform using your university login and password.
2. register to the INFO group and select the LPDIM courses.
3. select your previous diploma.
4. validate your registration.
5. at the next page, use key LPDIM to register to the DIM bachelor courses.
6. Find the test following this folder sequence : "Espace_cours_INFO-> INFO LP DIM -> AlgoC"
7. Do your best for your **unique attempt** and check your strength on basic algorithmic !

#1 Session 1 : Algorithm warm up and introduction to 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 *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.
  - 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.

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 python2.7 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 python2.7 language in the same script. *You must implement this algorithm as a function that follows prototype "function max_ value(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.

Reverse a table:

- Define one algorithm able to reverse a table **without the use of any other table**.
- Translate this algorithm in the python2.7 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 python2.7 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 K cells with value $'X'$ while the others should remain empty.
- Translate this algorithm in the python2.7 language in the same script. *You must implement this algorithm as a function that follows prototype : "function random_ fill_ sparse(table:numpy array, vfill) 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 python2.7 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 python2.7 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 python2.7 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 python2.7 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.