# Fouille de données – TD 1

Ce premier TD/TP a pour objectif de vous familiariser avec Python ainsi qu’avec des notions de base de statistiques. Lorsque l’on découvre ce langage très permissif, il est conseillé de lire ou de se référer au [Python Style Guide](https://google.github.io/styleguide/pyguide.html) qui donne les bonnes pratiques liées au code en Python.

**Il faudra rendre ce TD** sur [Moodle](https://moodle.u-paris.fr/course/view.php?id=11043) : rendez juste le fichier **td1.py**.

Testez votre code, exo par exo!

Téléchargez <http://fabien.viger.free.fr/ml/test.py> et <http://fabien.viger.free.fr/ml/test1.py> dans le même répertoire et executez: **python3 test1.py**

#### Exercice 1 – Listes, dictionnaires, ensembles

1. Écrivez une fonction average(lst) qui calcule la moyenne d’une liste de nombres lst. Vous pouvez dans un premier temps utiliser les fonctions len et sum de Python. Pouvez-vous ensuite modifier votre fonction pour ne pas les utiliser ?  
   Par exemple average([3, 7, 1, 2, 3]) doit retourner 3.2.

| def average(lst):  """ Computes the average of the values in a list of integers.  Args:  lst: a list of integers.  Returns:  A float that represents the average of the values in lst.  """  … |
| --- |

1. Écrivez une fonction median(lst) qui calcule la médiane d’une liste lst.  
   La médiane d’un ensemble est une valeur m telle qu’au moins une moitié de l’ensemble est ≤ m et au moins une moitié est ≥ m. Exemples: median([0, 8, 9]) **doit** retourner 8,  
   et median([7, 2, 3, 10, 3, 30]) **peut** retourner 5 (ou 6, ou 3.2, ou 3, ou 7, ou n’importe quel nombre entre 3 et 7 inclus)

| def median(lst):  """ Computes the median of the values in a list of integers.  Args:  lst: a list of integers.  Returns:  A float that represents the median of the values in lst.  """  … |
| --- |

Quelle est la complexité de cet algorithme? Comparez les résultats d’average et de  
 median appelées sur la liste [2, 4, 5, 6, 173]. Dans quelle(s) situation(s) de la vie  
 courante vous semble-t-il intéressant d’avoir recours à la médiane et non à la moyenne?

1. **[\*]** Écrivez une fonction bad\_sample(size, delta) qui renvoie une liste L de taille size telle que average(L) - median(L) = delta (attention, delta peut être négatif). Je vous laisse voir comment faire, il y a plein de manières!

| def bad\_sample(size, delta):  """ Returns a list of the given size, whose average = median + delta.  Args:  size: an integer ≥ 2, the size of the list to return.  delta: a number, the difference between the average and the median.  Returns:  A list of numbers of the given size, such that its average and median verify  average = median + delta.  """  … |
| --- |

1. **[\*]** Écrivez une fonction bad\_sample2(size, med, avg) qui renvoie une liste L de taille size telle que average(L) = avg et median(L) = med.

| def bad\_sample2(size, avg, med):  """ Returns a list of the given size, average and median.  Args:  size: an integer ≥ 2, the size of the list to return.  avg: a number, the average of the list to return.  med: a number, the median of the list to return.  Returns:  A list of numbers of the given size, average and median.  """  … |
| --- |

1. Écrivez une fonction occurrences(lst) qui retourne le nombre d’occurrences de chaque élément d’une liste lst sous forme de dictionnaire.  
   Par exemple l’appel occurrences([1, 2, 4, 3, 4, 1, 1, 3]) doit retourner le dictionnaire {1: 3, 2: 1, 3: 2, 4: 2} car 1 apparaît 3 fois, 2 apparaît 1 fois, etc.

| def occurrences(lst):  """ Computes the occurrences of the values in a list of integers.  Args:  lst: a list of integers.  Returns:  A dictionary where each key is a value in lst and each value is the number  of times the key appears in lst.  """  … |
| --- |

1. Écrivez une fonction unique(lst) qui retourne une liste contenant les éléments uniques de lst, **dans leur ordre d’apparition**.  
   Par exemple unique([2, 3, 1, 1, 3, 1, 4, 1, 3]) doit retourner [2, 3, 1, 4].

| def unique(lst):  """ Returns the unique values of a list of integers.  Args:  lst: a list of integers.  Returns:  A list containing the unique values in lst, in the same order as in the input  """  … |
| --- |

Quelle est la complexité de votre algorithme ? Pouvez-vous l’améliorer ?

#### Exercice 2 – Compréhension de listes

1. Écrivez une fonction squares(lst) qui retourne une liste contenant les carrés des éléments de lst.  
   Par exemple squares([1, 2, 3, 4]) doit retourner [1, 4, 9, 16].

| def squares(lst):  """ Squares the values of a list of integers.  Args:  lst: a list of integers.  Returns:  A list containing the squared elements of lst, in the same order.  """  … |
| --- |

1. Écrivez une fonction stddev(lst) qui retourne l’écart-type d’une liste lst.  
   stddev([5, 4, 7, 4, 4, 2, 5, 9]) doit retourner 2.0 et stddev([20, 20, 20]) doit retourner 0.0. Vous pouvez utiliser les fonctions des exercices précédent.

| def stddev(lst):  """ Returns the standard deviation of a list of integers.  Args:  lst: a list of integers.  Returns:  A float representing the empirical standard deviation of elements in lst.  """  … |
| --- |

1. (optionnel) Écrivez une fonction quicksort(lst) qui retourne la version triée de la liste lst en utilisant l’algorithme du tri rapide (*Quicksort*).  
   Pouvez-vous l’écrire sans boucle for ?

| def quicksort(lst):  """ Returns a sorted version of lst.  Args:  lst: a list of integers.  Returns:  A list of the integers from lst, in increasing order.  """  … |
| --- |

#### Exercice 3 – Simulation de variables aléatoires

Dans tout cet exercice, vous ne pouvez importer du module random que la fonction random, un générateur de nombres aléatoires entre 0 et 1 (exclus). L’en-tête de votre programme ne peut donc contenir que: from random import random (sans compter les autres, eg. import math)

Attention: les fonctions suivantes renvoient des variables aléatoires. Il est donc tout à fait normal (l’inverse serait inquiétant !) qu’elles renvoient des valeurs différentes à chaque appel.

1. Écrivez une fonction uniform() qui ne prend rien en entrée et renvoie une variable uniforme discrète: 0 ou 1, où chacune de ces valeurs a une probabilité 0.5 d’être renvoyée. Comment testeriez-vous votre fonction ?

| def uniform():  """ Returns a discrete uniform variable between 0 and 1, i.e., it returns 0  with probability 0.5 and 1 with probability 0.5.  """  … |
| --- |

1. **Modifiez** à présent cette fonction uniform() pour qu’elle retourne un nombre au hasard parmi 0, 1, 2, 3, 4, 5 (avec exactement ⅙ de chances chacun).

| def uniform():  """ Returns a discrete uniform variable between 0 and 5, i.e., the probability  of obtaining 0, 1, 2, 3, 4 or 5 is always 1/6.  """  … |
| --- |

1. **Modifiez** une dernière fois la fonction uniform(n) en lui donnant un argument n et pour qu’elle retourne un nombre parmi 0, 1, …., n-1.

| def uniform(n):  """ Returns a discrete uniform variable between 0 and n-1, i.e., it returns  every number between 0 and n-1 with the same probability.  Args:  n: a positive integer  """  … |
| --- |

1. Écrivez une fonction exam\_success(n, p) qui prend en entrée un entier n et un float p compris entre 0.0 et 1.0. n représente le nombre de cours pris par un étudiant, p représente la probabilité de valider un examen sans avoir rien révisé. Tous les cours sont indépendants. La fonction doit renvoyer le nombre d’examens réussis par un étudiant.

| def exam\_success(n, p):  """ Returns the number of exams passed by a student.  Args:  n: number of independent exams the students sits for.  p: probability of passing one exam.  Returns:  An integer representing the total number of exams passed by the  student.  """  … |
| --- |

#### Exercice 4 – Paradoxe de Monty Hall

1. Écrivez une fonction monty\_hall(change) qui simule un jeu de Monty Hall. Elle prend en argument un booléen change qui indique si le candidat décide de changer de porte, et retourne un booléen qui indique s’il a gagné. Vous pouvez utiliser la fonction randint du module random pour simuler le choix aléatoire d’une porte.

| def monty\_hall(change):  """ Simulates a Monty Hall game: a candidate is asked to choose between three  doors with only one of them containing a reward. Once the candidate has chosen  a door, the anchorman opens one of the other doors that does not contain the  reward. The candidate may now change doors for the remaining one. This function  returns 1 if the candidate found the reward and 0 otherwise.  Args:  change: a boolean representing whether the candidate changes doors.  Returns:  1 if the candidate found the reward at the end of the game, 0 otherwise.  """  … |
| --- |

1. Écrivez une fonction monty\_hall\_simulation(n) qui teste n jeux de Monty Hall avec et sans changement de porte, et affiche la fréquence des victoires pour chaque cas. Que pouvez-vous déduire du résultat ?

| def monty\_hall\_simulation(n):  """ Simulates n Monty Hall games with the candidate changing doors and n Monty  Hall games with the candidate not changing doors. Returns the frequency with  which the candidate found the reward in each case.  Args:  n: the number of games to simulate.  Returns:  A tuple (p1, p2) where p1 (resp. p2) is the frequency with which the  candidate found the reward when always (resp. never) changing doors.  """  … |
| --- |