

TP n° 7 : Quelques algorithmes autour des listes

I Étude statistique

1 Minimum, maximum, classes

On se donne une série de données statistiques, sous forme d'une liste de nombres compris entre **inf** et **sup**. On considère la série statistique suivante qui contient les notes de la première interrogation de S2I en MPSI :

```
donneestat=[7, 15, 11, 15, 15, 12, ... ]
```

Script 1 ▼

Écrire une fonction **verif** qui teste si tous les éléments d'une série statistique reçue sous la forme d'une liste sont bien compris entre les valeurs **inf** et **sup**, deux arguments passés aussi comme arguments de **verif**. La fonction **verif** reçoit donc trois arguments.

Application ► Vérifier que toutes les notes de l'interrogation de S2I sont comprises entre 0 et 20.

Remarque : c'est le problème des interrogations de cours de type QCM avec points négatifs sur les mauvaises réponses, il peut y avoir des notes négatives.

Script 2 ▼

Écrire la fonction **modif_serie**, qui aura en arguments d'entrée, une liste de nombre, une valeur **inf** et une valeur **sup** à ne pas dépasser. Cette fonction remplace les éléments de la série inférieurs à **inf** par la valeur **inf** et les éléments supérieurs à **sup** par la valeur **sup**.

Application ► Utiliser cette fonction pour avoir des notes comprises entre 0 et 20. Vérifier le résultat.

Étant donné un segment $[a, b]$ et un nombre N , on appelle **classes** les N sous-intervalles de $[a, b]$ qui permettent de le découper en N parties de même longueur. Cette longueur est donc $\alpha = \frac{b-a}{N}$.

Plus précisément les N classes sont données par :

$$\begin{aligned} &[a, a + \alpha[, \quad [a + \alpha, a + 2\alpha[, \quad [a + 2\alpha, a + 3\alpha[, \dots \\ &[a + (N-2)\alpha, a + (N-1)\alpha[\quad \text{et enfin} \\ &[a + (N-1)\alpha, a + N\alpha] = [b - \alpha, b]. \end{aligned}$$

Script 3 ▼

Écrire une fonction Python ayant pour arguments les nombres de la série statistique sous forme d'une liste, les valeurs extrêmes a et b à considérer et le nombre N de classes. Et dont le résultat est la liste des effectifs de chacune des N classes.

Application ► Quel est l'effectif de chacune des classes de `donneestat` si N vaut 4 ?

2 Effectifs cumulés

L'effectif cumulé (croissant) d'une valeur (ici donnée statistique) est le nombre d'individus dont le caractère a une valeur inférieure ou égale à celle-ci. Dans notre cas, on se donnera une liste de données statistiques et N le nombre de classes à considérer et l'intervalle $[a, b]$ utilisé pour fabriquer les classes. On ne considérera ensuite que la répartition obtenue en terme de classes.

Script 4 ▼

Écrire une fonction Python qui détermine la listes des effectifs cumulés (pour la répartition en classe) d'une série statistique donnée sous forme d'une liste.

Application ► Quel est le résultat des effectifs cumulés de la liste `donneestat` ?

3 Médiane

On appelle **médiane** d'une série statistique toute valeur m qui sépare les données de manière à ce 50% des données sont supérieures à m et 50% des données sont inférieures à m . La **classe médiane** d'une série statistique sera donc la classe la plus petite dont les effectifs cumulés dépassent strictement 50% de l'effectif total.

Script 5 ▼

Écrire une fonction Python qui détermine la classe médiane (en donnant ses bornes) d'une série statistique que l'on répartira préalablement en N classes sur l'intervalle $[a; b]$.

Application ► Quelle est la médiane de la liste `donneestat` ?

II Crible d'Ératosthène

Le crible d'Ératosthène est une méthode permettant de déterminer tous les nombres premiers situés entre 0 et n avec $n > 2$ un entier naturel donné (par exemple $n = 100$).

À la main, on procède ainsi : on écrit sur une feuille tous les nombres compris 0 à n , dans l'ordre croissant et on élimine progressivement tous les nombres qui ne sont pas premiers par le procédé suivant :

- On raye d'abord les nombres 0 et 1 car ils ne sont pas premiers.
- 2 est premier : on le conserve. En revanche, on raye tous les multiples de 2 car ils ne sont pas premiers : 4, 6, 8 et ainsi de suite jusqu'à atteindre n sont éliminés.
- Le premier nombre non rayé après 2 est le nombre 3. Il s'agit du deuxième nombre premier détecté. On raye ses multiples : 6 (déjà rayé à l'étape précédente), 9, 12 (déjà rayé) etc. jusqu'à n .
- Le premier nombre non rayé est maintenant 5 : il est premier. On raye ses multiples : 10, 15, 20 (déjà rayés aux étapes précédentes), 25 (non rayé) etc.

... et ainsi de suite jusqu'à ce qu'on ait épuisé tous les nombres écrits au départ. À la fin, les nombres qui n'ont pas été rayés sont les nombres premiers inférieurs ou égaux à n .

Remarque Quand on vient de trouver un nombre premier p et qu'on s'appête à rayer ses multiples, le premier multiple de p non rayé lors des étapes précédentes est p^2 .

Question 6 ▼

Assurez-vous que vous avez compris la méthode en l'appliquant sur un papier pour déterminer tous les nombres premiers jusqu'à 50.

Script 7 ▼

Écrire une fonction python `crible` permettant de déterminer la liste des nombres premiers inférieurs à un entier $n > 2$ donné.

Indications

- On commencera par fabriquer une liste `raye` contenant $(n + 1)$ éléments tous égaux à `False`. Pour « rayer le nombre k », on remplace l'élément `raye[k]` par `True`.
- À la fin de l'algorithme, la liste `raye` contiendra des `False` et des `True`. Les nombres premiers seront les indices k pour lesquels `raye[k]` sera resté à `False`.
- À partir du tableau `raye` final, on construira une liste contenant uniquement les nombres premiers inférieurs ou égaux à n .

Exemple d'utilisation

```
In : crible(20)
Out: [2, 3, 5, 7, 11, 13, 17, 19]
```

Script 8 ▼

Écrire un algorithme (puis sa traduction Python sous forme d'une fonction `decompose`) qui, à partir d'un entier n , fournit la décomposition en nombres premiers de n . La réponse sera fournie sous forme d'une liste de couple $[p, e]$ où p est un nombre premier intervenant dans la décomposition de n , et e l'exposant de p dans cette décomposition.

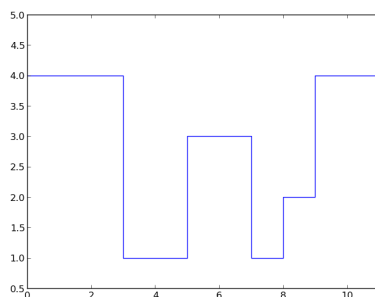
Exemple d'utilisation, avec $2268 = 2^2 \times 3^4 \times 7$:

```
In : decompose(2268)
Out: [[2, 2], [3, 4], [7, 1]]
```

Application ► En déduire la décomposition en nombre premiers de 1224.

III Les plateaux

On se donne une liste de valeurs (on pourra y penser comme une suite d'altitudes). Un plateau est une succession de cases consécutives contenant la même valeur (à la même altitude). Par exemple, `tab = [4, 4, 4, 1, 1, 3, 3, 1, 2, 4, 4]` est un tableau comportant 6 plateaux (cf figure ci-dessous). Le premier vaut 4 et a pour longueur 3, le cinquième vaut 2 et a pour longueur 1.



On peut décrire la liste (des altitudes) à l'aide de ses plateaux. La liste compressée est alors une liste de listes à deux éléments [valeur, longueur_du_plateau] qui permet de décrire la succession des plateaux. Dans le cas de *tab* (donnée précédemment), la liste compressée correspondante est *tab_comp* = [[4, 3], [1, 2], [3, 2], [1, 1], [2, 1], [4, 2]].

Script 9 ▼

Écrire une fonction python *decode* qui reçoit en argument une liste *description*, sous la forme d'une liste de listes à deux éléments [valeur, longueur_du_plateau], et qui fabrique la liste décrite par *description* (i.e. *description* est la liste cherchée sous forme compressée).

Par exemple, avec l'exemple, *decode(tab_comp)* renverra la liste *tab*.

Script 10 ▼

Écrire une fonction python *nb_plateau* qui détermine le nombre de plateaux d'une liste de nombres (correspondant à une liste des altitudes) passée en argument.

On pourra penser à détecter un changement de plateau en comparant les valeurs de la liste prises sur deux indices successifs.

Script 11 ▼

Écrire une fonction python qui détermine la longueur du plus grand plateau d'une liste de nombres passée en argument.

Script 12 ▼

Écrire une fonction python *comprime* qui renvoie un tableau récapitulant la valeur et la longueur des plateaux successifs du tableau passé en argument, autrement dit la fonction *comprime* renvoie la version codée (sous forme d'une liste de listes à 2 éléments) de la liste de nombres passée en arguments.