Séance 8: Opérations sur les listes d'entiers

Université de Paris Cité

Objectifs:

— Manipuler des listes unidimensionnels d'entiers.

Dans cette séance, vous résoudrez des exercices et des problèmes sur des listes d'entiers. Vous écrirez des boucles et définirez des fonctions intermédiaires pour rendre votre code plus lisible et concis.

Exercice 1 (Test liste triée, ⋆)
On dit qu'une liste d'entiers est triée en ordre croissant *si l'entier contenu dans la case d'indice* `i` *est inférieur ou égal à l'entier contenu dans la case d'indice* `i + 1`*, pour tout indice.*

Écrire une fonction `isSorted(lis)` *qui renvoie* `True` *si la liste* `lis` *est triée en ordre croissant, et* `False` *sinon.*

Contrat:
`isSorted([1,0,4,5])` *renvoie* `False`
`isSorted([0,1,5,4])` *renvoie* `False`
`isSorted([0,1,4,5])`, `isSorted([])` *et* `isSorted([7])` *renvoient tous les trois* `True`

□

Exercice 2 (Supprimer les zéros, ⋆)
Écrire une fonction `compress` *qui prend en paramètre une liste d'entiers* `lis` *et qui renvoie la liste obtenue à partir de* `lis` *en supprimant tous les* `0`*. En effet, avec les notions que vous connaissez de* Python*, vous ne pouvez pas supprimer des éléments dans une liste. Ce que nous allons faire, c'est créer une nouvelle liste dans laquelle nous mettons seulement les éléments souhaités.*

Contrat:
`compress([0,3,-4,0,0,7])` *renvoie* `[3,-4,7]`.

□

Exercice 3 (Maximum, ⋆)
Écrire une fonction `maximum` *qui prend en argument une liste d'entiers* `lis` *et qui renvoie l'indice* `i` *du maximum de cette liste, ou* `-1` *si* `lis` *est vide. Si le maximum est atteint plusieurs fois, c'est-à-dire, s'il y a plusieurs* `i` *tels que* `lis[i]` *est la valeur maximale,* `maximum` *a le droit de renvoyer n'importe lequel de ces* `i`*.*

Contrat:
`maximum([])` *renvoie* `-1`
`maximum([1000,1,1,1])` *renvoie* `0`.
`maximum([0,5,1,2,5])` *renvoie* `1` *ou* `4`.

□

Exercice 4 (Signes, ⋆)
Le signe *d'un entier n est 0 si $n = 0$, 1 si $n > 0$ et -1 si $n < 0$. Écrire une fonction* `samesign` *qui prend en*

argument deux listes d'entiers `l1` *et* `l2` *et qui renvoie la liste* `lis` *dont la valeur en la coordonnée* `i` *vaut* `1` *si les deux listes* `l1` *et* `l2` *ont des valeurs de même signe en la coordonnée* `i`, `-1` *sinon. Si les tailles des listes sont différentes la fonction renvoie la liste* `[2]`.

Contrat:
`l1=[1000,0,-3,1], l2=[-1,0,-1,-1000]`
`samesign(l1,l2)` *renvoie* `[-1,1,1,-1]`.

□

Exercice 5 (Décalage, ⋆⋆)
Écrire une fonction `shift` *qui prend en argument une liste d'entiers* `lis` *et qui renvoie une copie de cette liste où toutes les valeurs sont décalées d'une case vers la droite (et la dernière valeur se retrouve en position 0).*

Contrat:
`lis=[1000,1,2,3]`
`shift(lis)` *renvoie* `[3,1000,1,2]`.

Même chose mais avec un décalage de `n` *cases (*`n` *est un paramètre). Que faire quand* `n` *est négatif ?* □

Exercice 6 (La liste des nombres premiers, ⋆⋆⋆)
Dans cet exercice, nous écrirons une fonction calculant la liste de tous les nombres premiers plus petits qu'un certain nombre donné en entrée. Pour ce faire, on utilisera une méthode connue sous le nom de crible d'Ératosthène, *qui consiste à déterminer les nombres premiers par filtrages successifs. Imaginons d'être sur un chemin infini éclairé par une suite de lampadaires numérotés à partir de 2, tous allumés. À côté de chaque lampadaire k, il y a un interrupteur qui a l'éffet d'éteindre tous les lampadaires numérotés par un multiple de k, sauf k lui-même. On commence à se promener le long du chemin et, à chaque fois que l'on passe à côté d'un lampadaire allumé, on appuie sur l'interrupteur correspondant. Au fur et à mesure, les seuls lampadaires allumés que l'on laissera derrière nous seront exactement ceux correspondants aux nombres premiers.*

1. *Écrire une fonction* `sieve` *ayant comme paramètres une liste* `lis` *et un entier* `k` *(que l'on suppose positif) et renvoyant la liste obtenue à partir de* `lis` *en mettant tous les éléments d'indice multiple de* `k` *à* `0`, *sauf l'indice* `k` *lui-même.*

   Contrat:
   `sieve([0,0,2,3,4,5,6,7,8,9,10], 3)` *renvoie [0,0,2,3,4,5,0,7,8,0,10].*

2. *Écrire une fonction* `EratoSieve(n)` *qui renvoie la liste* `[0,0,2,3,4,...,n-1]` *à laquelle on a appliqué le crible d'Ératosthène (attention, il y a un* `0` *en position* `1` *: par définition, 1 n'est pas premier !). De cette façon,* `compress(EratoSieve(n))`, *où* `compress` *est la fonction de l'Exercice 2, calculera la liste des nombres premiers strictement plus petits que* `n`, *comme voulu.*

   Contrat:
   `EratoSieve(10)` *renvoie* `[0,0,2,3,0,5,0,7,0,0]`.
   `compress(EratoSieve(100))` *renvoie* `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]`.
   `len(compress(EratoSieve(100000)))` *renvoie* 9592.[1]

□

1. De manière générale, `len(compress(EratoSieve(n)))` renvoie le nombre de nombres premiers strictement inférieurs à `n`. Estimer de façon aussi précise que possible ce nombre, dénoté habituellement $\pi(n)$, est un des problèmes centraux de la branche des mathématiques appelée *théorie des nombres*. Par exemple, si vous arrivez à démontrer que $\left|\pi(n) - \int_0^n \frac{dt}{\log t}\right| < \sqrt{n}\log n$ pour tout $n \geq 2$, vous aurez atteint la gloire éternelle (ainsi que gagné 1 million de dollars). Du point de vue numérique, nous avons vu comment calculer "rapidement" $\pi(n)$ pour n de l'ordre de millions (essayez `len(compress(EratoSieve(1000000)))` et regardez combien de temps cela prend). Pour aller au delà, il faut des méthodes plus sophistiquées.