Exercices de Programmation & Algorithmique 1

Série 5 – Les listes
(20 octobre 2022)

Département d'Informatique – Faculté des Sciences – UMONS

Pré-requis : Listes Python : opérateurs, méthodes et opérations fréquentes sur les listes; listes et chaînes de caractères, introduction à la notion d'objets (cours jusqu'au **Chapitre 7** compris).

Objectifs : Être capable d'utiliser et de manipuler des séquences de valeurs via les listes; concevoir des algorithmes sur des séquences dynamiques de valeurs (plutôt qu'en utilisant un nombre constant de variables)

---

1 Le contrat

1.1 Le contexte

Et si vous étiez à la place des développeurs Python et que la librairie standard ne possédait presque pas de méthodes ou de fonctions sur les listes ? Cette série d'exercices va vous permettre d'imaginer comment vous auriez pu faire pour concevoir certains des algorithmes offerts par la librairie standard.

Dans ce but, la contrainte suivante vous est imposée pour remplir votre contrat et réaliser les exercices complémentaires : **les seules méthodes des listes Python que vous pouvez utiliser sont les méthodes `append` et `pop`. Vous pouvez également utiliser l'opérateur `[.]` et ses variantes** (*slice* par exemple). Certains exercices ne tiennent pas compte de cette contrainte, ce qui sera clairement indiqué dans l'énoncé. **Si rien n'est précisé, la contrainte est d'application**. Par contre, dans vos tests, vous pouvez librement utiliser les opérateurs et fonctions pour créer des itérateurs et des listes (`list(range([...]))` par exemple).

1.2 A réaliser sur machine

Créez un module `myList` qui contiendra le code des fonctions demandées ci-dessous.

1 Trois opérations très fréquentes sur les listes ont été vues au cours (chapitre 7) : `map`, `filter` et `reduce`. Dans le module `myList`, ajoutez le code de 3 fonctions réalisant ces opérations sur les listes. Attention : ces opérations ne modifient pas la liste passée en paramètre mais retournent une nouvelle liste (ou une valeur dans le cas de `reduce`). Pour la fonction `reduce`, on considère (précondition) que la liste contient au moins deux éléments et vous ne devez pas gérer le paramètre optionnel `initial`.

Tests :

```
myList.map(math.sqrt, [])  →  []
```

```
myList.map(math.sqrt, [2.0, 4.0, 6.0, 100.0])
→  [1.4142135623730951, 2.0, 2.449489742783178, 10.0]
```

```
myList.map(str.upper, list('hello'))  →  ['H', 'E', 'L', 'L', 'O']
```

```
myList.filter(myList.is_prime, range(20))  →  [2, 3, 5, 7, 11, 13, 17, 19]
```

```
myList.filter(str.isalpha, list('r2d2'))  →  ['r', 'd']
```

```
myList.reduce(math.pow, [2, 2])  →  4.0
```

```
myList.reduce(math.pow, [2, 3, 4])  →  4096.0
```

2 Ajoutez au module `myList` l'algorithme `prime_numbers(n)`. Vous ne pouvez pas utiliser la fonction `is_prime(n)` de l'exercice suivant.

Entrée : un nombre entier n.

Sortie : une liste contenant les n premiers nombres premiers.

Note : pour rappel, on peut tester si un nombre i est premier en vérifiant qu'il n'est divisible par aucun des nombres premiers plus petits ou égaux à $\sqrt{i}$ (cf. cours de Math. élem.). **Exploitez au mieux cette propriété dans votre algorithme.** Les nombres 0 et 1 ne sont pas premiers.

Tests :

```
myList.prime_numbers(1)  →  [2]
myList.prime_numbers(12)  →  [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

3 Ajoutez au module `myList` l'algorithme `is_prime(n)`.

Entrée : un nombre entier n.

Sortie : un booléen qui est vrai si et seulement si n est un nombre premier.

Tests :

```
myList.is_prime(1)  →  False
myList.is_prime(2)  →  True
myList.is_prime(3)  →  True
myList.is_prime(33)  →  False
```

4 Ajoutez au module `myList` l'algorithme `insert(seq, n)`.

Précondition : les éléments de la liste `seq` sont triés par ordre croissant.

Entrée : une liste L contenant des entiers et un nombre entier n.

Sortie : rien mais la liste `seq` a été modifiée en insérant n au bon endroit pour que celle-ci reste triée.

Tests :

```
myList.insert([], 1)  →  [1]
myList.insert(list(range(6)), -1)  →  [-1, 0, 1, 2, 3, 4, 5]
myList.insert(list(range(6)), 3)  →  [0, 1, 2, 3, 3, 4, 5]
myList.insert(list(range(6)), 10)  →  [0, 1, 2, 3, 4, 5, 10]
```

5 Écrivez une fonction `produit_matriciel` qui permet de calculer le produit de deux matrices A et B telles que $A = (a_{ij})$ est une matrice réelle de taille $m \times n$ (m lignes et n colonnes) et $B = (b_{ij})$ est une matrice réelle de taille $n \times p$. Pour rappel, le produit matriciel de A et B est une matrice $C = (c_{ij})$ de taille $m \times p$ telle que chaque élément c_{ij} est défini par

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}.$$

Une matrice $m \times n$ sera représentée en Python par une liste (de longueur m) de listes (de longueur n). Par exemple, la matrice 3×2 suivante

$$\begin{pmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{pmatrix}$$

sera représentée par `[[1, 5], [2, 6], [3, 7]]`. Pour une matrice en entrée, on suppose que chaque ligne a la même longueur (*précondition*). Par contre, vous devez tester si le nombre de colonnes de la première matrice correspond au nombre de lignes de la seconde matrice. Dans le cas contraire, votre fonction retournera `None`.

Tests :

```
produit_matriciel([[2, 0, 1], [3, 6, 2]], [[1, 5], [2, 6], [3, 7]])
→  [[5, 17], [21, 65]]
produit_matriciel([[1, 5], [2, 6], [3, 7]], [[2, 0, 1], [3, 6, 2]])
→  [[17, 30, 11], [22, 36, 14], [27, 42, 17]]
produit_matriciel([[1.0, 2.5]], [[3.0], [4.5]])  →  14.25
produit_matriciel([[1.0, 2.5]], [[3.0, 4.5]])  →  None
produit_matriciel([[1, 5], [2, 6], [3, 7]], [[1, 5], [2, 6], [3, 7]])
→  None
```

Note : pas de contrainte sur les méthodes et fonctions utilisées.

2 Exercices supplémentaires

★☆☆ 6 Ajoutez au module `myList` la fonction `count(liste, elem)`.

Entrée : une liste `liste` et un élément `elem`.

Sortie : le nombre de fois que `elem` est présent dans `liste`.

Tests :

```
myList.count(list(range(5)) * 4, 3)  →  4
```

```
myList.count(range(6), 10)  →  0
```

★☆☆ 7 Ajoutez au module `myList` la fonction `pow(nb, base)`. **Entrée :** un nombre entier `nb` et un nombre flottant `base`.

Sortie : une liste contenant les `nb`-èmes premières puissances de `base` :

$$[base^0, base^1, base^2, ..., base^{nb-1}]$$

Note : vous ne pouvez utiliser ni l'opérateur `**`, ni la fonction `math.pow`.

Tests :

```
myList.pow(3, 2)  →  [1, 2, 4]
```

```
myList.pow(7, 1.5)  →  [1, 1.5, 2.25, 3.375, 5.0625, 7.59375, 11.390625]
```

★★☆ 8 Ajoutez au module `myList` la fonction `remove(liste, elem)`.

Entrée : une liste `liste` et un élément `elem`.

Sortie : néant, mais la liste passée en paramètre est modifiée : la fonction supprime la première occurrence de `elem` dans `liste`, le cas échéant.

Tests :

```
myList.remove(list(range(5)), 3)  →  [0, 1, 2, 4]
```

```
myList.remove([1, 2, 3, 2], 2)  →  [1, 3, 2]
```

```
myList.remove([1, 2, 3, 2], 5)  →  [1, 2, 3, 2]
```

★★☆ 9 Ajoutez au module `myList` la fonction `print1(liste, separator)`.

Entrée : une liste `liste` et un tuple (triplet) `separator`.

Sortie : une chaîne de caractère contenant dans l'ordre :

(a) le premier élément du `separator` ;

(b) chaque élément de `liste` en utilisant comme séparateur le deuxième élément du `separator` ;

(c) le troisième élément du `separator`.

Tests :

```
liste = [1,2,3,4,5]
separator = ('[ ',' -> ',' ]')
print(myList.print1(liste, separator))  →  [ 1 -> 2 -> 3 -> 4 -> 5 ]
```

★★★ 10 Ajoutez au module `myList` la fonction `print2(liste, separators)`.

Entrée : une liste `liste` et une liste `separators` de tuples (triplets).

Sortie : une chaîne de caractère comme décrite ci-après. Le $i^{\text{ème}}$ tuple de `separators` sert à séparer lors de l'affichage (comme le fait `print1`) les éléments de $i^{\text{ème}}$ profondeur de `liste`. S'il n'y a pas de séparateur pour un niveau de profondeur de la liste (si par exemple, la liste est de profondeur 3, mais qu'il n'y a que 2 tuples séparateurs), les éléments de profondeurs 3 sont uniquement concaténés.

Tests :

```
liste = [[0, 1, 2], [3, 4, 5, 6], [7, 8, 9, 10, 11]]
separators = [('*******\n', '\n', '\n*******'), ('[ ',' -> ',' ]')]
print(myList.print2( liste, separators))
```

→

```
*******
[ 0 -> 1 -> 2 ]
[ 3 -> 4 -> 5 -> 6 ]
[ 7 -> 8 -> 9 -> 10 -> 11 ]
*******
```

```
liste = [['x','y'], ['z','p'], ['m',['a','b','c',['d','e']]]]
separators = [('{', ' = ', '.'), ('f(', ',', ')'), ('[', ' + ', ']'), ('','*','')]
print(myList.print2(liste,separators))
```

→

```
{f(x,y) = f(z,p) = f(m,[a + b + c + d*e]).
```

★☆☆ 11 Ecrivez une fonction `somme_totale(A)` qui retourne la somme des valeurs d'une matrice représentée comme à l'exercice 5.

★☆☆ 12 Ecrivez une fonction `somme_matricielle(A, B)` qui retourne la matrice étant la somme de deux matrices A et B représentées comme à l'exercice 5 (et qui retourne `None` si les dimensions ne sont pas valides).

★★☆ 13 Ecrivez une fonction `transposee(A)` qui retourne la transposée d'une matrice représentée comme à l'exercice 5.

Exam 14 (30 août 2010) Problème 4 : régression linéaire.

Exam 15 (3 novembre 2010) Problème 3 : nombre premiers.

Exam 16 (3 novembre 2010) Problème 4 : opérations matricielles.

Exam 17 (18 janvier 2011) Problème 1 : recherche dichotomique.

Exam 18 (10 juin 2011) Problème 1 : déterminant d'une matrice.

Exam 19 (10 juin 2011) Problème 3 : compression de chaînes.

Exam 20 (26 octobre 2011) Problème 1 : matrice des sommes progressives.

Exam 21 (26 octobre 2011) Problème 4 : codage de Fibonacci.

Exam 22 (13 juin 2012) Problème 3 : recherche dichotomique.

Devoir 23 (novembre 2009) Projet sur les matrices

Devoir 24 (novembre 2010) Projet de manipulation d'images