Récursivité

Pour débugger votre code, n'hésitez pas a utiliser PythonTutor :
https://pythontutor.com/render.html#mode=edit

Exercice 1 :

```
# Somme des entiers de 0 à n
def somme(a : int) -> int :
     if a == 0:
          return 0
     else:
          return a +  somme(a-1)

print(somme(10)) # 55
```

En prenant exemple de la fonction `somme` :
1) Écrire la fonction récursive `factoriel(n : int) -> int` qui renvoie $n!$.
2) Écrire la fonction récursive `produit(a : int, b : int) -> int` qui renvoie $a \times b$.
3) Écrire la fonction récursive `puissance(a : int, b : int) -> int` qui renvoie a^b.
4) Écrire la fonction récursive `somme(a : int, b : int) -> int` qui renvoie la somme des entiers de `a` à `b`.

Exercice 2 :

Slicing
Le slicing permet d'obtenir une sous liste qui contient une copie des éléments de la liste de départ.
La Syntaxe est la suivante : `list_name[start : end : step]`
Tous les paramètres sont optionnels.

```
l = [0, 1, 10, 11, 100]
l[2:]     # [10, 11, 100]
l[:3]     # [0, 1, 10]
l[1:3]    # [1, 10]
l[::2]    # [0, 10, 100]
```

1) Écrire la fonction récursive `somme(l : list[int]) -> int` qui renvoie la somme des éléments de `l`. On utilisera le slicing.
2) Écrire la fonction récursive `somme(l : list[int], indice: int = 0) -> int` qui renvoie la somme des éléments de `l`. Utiliser le paramètre `indice` pour parcourir la liste.

3) Écrire la fonction récursive `minimum(l : list[int]) -> int` qui renvoie le plus petit élement de `l`. On utilisera le slicing.
4) Écrire la fonction récursive `minimum(l : list[int], indice : int = 0) -> int` qui renvoie le plus petit élement de `l`. Utiliser le paramètre `indice` pour parcourir la liste.

Récursion terminale

Une fonction à récursivité terminale est une fonction où l'appel récursif est la dernière instruction à être évaluée. Cette instruction est alors nécessairement « pure », c'est-à-dire qu'elle consiste en un simple appel à la fonction, et jamais à un calcul ou une composition. Par exemple, dans un langage de programmation fictif :

```
def récursionTerminale(n) :
     # ...
     return récursionTerminale(n - 1)

def récursionNonTerminale(n) :
     # ...
     return n + récursionNonTerminale(n - 1)
```

`récursionNonTerminale()` n'est pas une récursion terminale car sa dernière instruction est une composition faisant intervenir l'appel récursif. Dans le premier cas, aucune référence aux résultats précédents n'est à conserver en mémoire, tandis que dans le second cas, tous les résultats intermédiaires doivent l'être. Les algorithmes récursifs exprimés à l'aide de fonctions à récursion terminale profitent donc d'une optimisation de la pile d'exécution.

```
# somme des entiers de 0 à n récursive terminale
def somme_terminal(n : int, acc : int = 0) -> int:
     if n == 0:
          return acc
     else:
          return somme_terminal(n - 1, acc + n)
print(somme_terminal(10)) # 55
```

Les différences entre la version récursive et la version récursive terminale :
- On utilise un paramètre `acc` (accumulateur) pour stoquer les résultats successifs des appels de fonction.
- La dernière instruction effectuée est toujours un simple appel de fonction.
- A la fin de la récursion, on retourne l'accumulateur.

Exercice 3 :

Réécrire les fonctions des exercices 1 et 2 avec de la récursion terminale.