
Dans un premier temps, on répondra aux questions à l'écrit puis pour vérifier que nos algorithmes sont bien écrits, on les programmera sur machine.
Sur papier, il n'est pas obligatoire d'écrire en Python et les erreurs de syntaxe sont acceptées.

Exercice 1 :

Écrire un algorithme qui calcule $n!$. On prouvera la terminaison et la correction de celui-ci.

Exercice 2 :

(1) Écrire une fonction **chercher** qui prend en argument une liste et un nombre et renvoie l'indice de ce nombre dans la liste si il existe et -1 sinon. Par exemple :

```
L = [1,2,5]
print (chercher(L,2))
  --> 1
print (chercher(L,3))
  --> -1
```

(2) Montrer la terminaison de cet algorithme.

(3) Exprimer un invariant de boucle pour la boucle de la fonction **chercher**. Utiliser cet invariant pour démontrer la validité de cette fonction.

Exercice 3 :

Écrire la fonction *moyenne(tab)* qui prend un tableau de nombre en paramètre et renvoie la moyenne des nombres.

Exercice 4 :

Soit *tab[0...n-1]* un tableau de n entiers, $n \geqslant 1$.

(1) On considère dans cette question que *tab* est un tableau trié jusqu'à l'indice $j-1$ ($j < n$).

a. Écrire la fonction **insere(tab, elem, j)** qui insère l'élément *elem* dans *tab[0..j-1]* en laissant le tableau trié jusqu'à l'indice j.

Par exemple :

```
L = [1, 3, 5]
print(insere(L, 2, 2))
--> [1, 2, 3]
L = [1, 4, 7]
print(inselere(L, 3, 1))
--> [1, 3, 7]
```

Solution:

```
def insere(tab, elem, ordre):
    i = ordre
    while i>0 and tab[i-1]>elem:
        tab[i] = tab[i-1]
        i = i-1
    tab[i] = elem
```

b. Montrer la terminaison de cet algorithme.

c. Exprimer un invariant de boucle pour démontrer la validité de la fonction **insere**.

> **Solution: Invariant de boucle** : À la fin de l'itération j, le tableau $tab[ordre - j + 1 : ordre]$ est trié et $elem < tab[ordre - j]$.
> **Initialisation** : Lorsque $j = 0$ le tableau est vide donc il est trié.
> **Conservation** : Supposons qu'à la fin de la $j$ème itération, le tableau $tab[ordre - j + 1 : ordre]$ est trié et $elem < tab[ordre - j]$ et que l'on fasse une nouvelle itération. Si on a fait une nouvelle itération c'est que $elem < tab[ordre - j - 1]$.
> Au début de cette nouvelle itération, on a donc $tab[ordre - j + 1] = tab[ordre - j]$. De plus, on sait que le tableau était trié à l'origine, on a donc $tab[ordre - j - 1] \leqslant tab[ordre - j]$.
> À la fin de l'itération $j + 1$, on effectue l'opération $tab[ordre - j] = tab[ordre - j - 1]$, on a donc $tab[ordre - j] \leqslant tab[ordre - j + 1]$. On en conclut donc que $tab[ordre - j : ordre]$ est trié.
> **Terminaison** À la sortie de boucle on sait que $tab[ordre - j : ordre]$ est trié et $elem \leqslant tab[ordre - j]$ avec $ordre = j$ ou $tab[ordre - j - 1] \geqslant elem$.
> — Si $ordre = j$, alors $tab[0] = elem$, $elem \leqslant tab[1]$ et $tab[1 : ordre]$ est trié donc la terminaison est prouvé.
> — Si $tab[ordre - j - 1] \geqslant elem$, alors $tab[ordre - j] = elem$ avec $tab[ordre - j] \leqslant tab[ordre - j + 1]$ et $tab[ordre - j + 1 : ordre]$ est trié. Comme $tab[0 : ordre - j - 1]$ était déjà trié, on en déduit la terminaison.

(2) Le tableau n'est plus trié.

On se donne la fonction **a_quoi_ca_sert** :

```
def a_quoi_ca_sert(tab):
    for j in range(1, len(tab)):
        insere(tab, tab[j], j)
```

a. Dire à quoi sert cette fonction.

b. Prouver la validité de celle-ci.

> **Solution:** À la fin de l'itération j, le tableau $tab[0 : j]$ est trié. **Initialisation** : Pour $j = 0$, le tableau a un seul élément donc il est trié. **Conservation** : Supposons qu'à la fin de l'itération j, $tab[0 : j]$ soit trié et que l'on fasse une nouvelle itération. En appelant *insere(tab, tab[j], j+1)*, on a le tableau $tab[0 : j + 1]$ trié avec les anciens éléments de '$tab[0 : j]$ plus $tab[j + 1]$. **Terminaison** En sortie de boucle, $j = n - 1$, donc on en déduit que le tableau tab est trié.