

Séance 6 : LISTES, GESTION MÉMOIRE ET EXCEPTIONS

L1 – Université Côte d'Azur

Exercice 1 — Gestion des absences

Dans cet exercice, on considère des listes de notes. On considère qu'une note est soit un flottant entre 0 et 20, soit la chaîne de caractères "ABS".

1. Écrire une procédure `modifier_ABS(liste)` qui modifie la liste donnée en argument en remplaçant les "ABS" par des zéros. Écrire un test avec `assert`.
2. Écrivez une fonction `ABS_vers_zéro_basique(liste)` qui à partir d'une liste de note renvoie un nouvelle liste obtenue en remplaçant les "ABS" par zéro. La liste initiale ne devra pas être modifiée et on ne pourra pas utiliser la méthode `append`. Écrire un test
3. Écrivez une fonction `ABS_vers_zéro(liste)` ayant le même comportement que la fonction précédente, mais cette fois-ci en utilisant `append`.
4. Enfin écrivez une fonction `filtrer_notes(liste)` qui renvoie la liste obtenue à partir de l'argument en supprimant tous les "ABS". La liste renvoyée sera donc plus courte que la précédente.
5. Aurait-on pu écrire cette fonction sans utiliser `append` (ni compréhensions).

Exercice 2 — Comprendre les compréhensions

Pour chacune des constructions de liste par compréhension, donnez la valeur finale de la liste puis écrivez une boucle `for` construisant une telle liste. On pourra utiliser la méthode `append`.

1. `[x*x for x in range(4)]`
2. `[(i,10-i) for i in range(11)]`
3. `[str(i) for i in range(11) if i%2==0]`

Exercice 3 — Maximisation filtrante

On considère dans cet exercice des listes d'entiers.

1. Écrivez une fonction `max_pair(liste)` renvoyant la plus grande valeur paire de la liste. Si la liste n'en contient aucune, on soulèvera une `ValueError`.

```
1  >>> max_pair([9, 17, 8, 11, 4])
2  8
3  >>> max_pair([9, 7, -12, 11])
4  -12
5  >>> max_pair([9, 7, 11])
6  Traceback (most recent call last):
7  File "<console>", line 1, in <module>
8  File "<console>", line 12, in max_pair
9  ValueError: Pas de valeur paire
10 >>> max_pair([])
11 Traceback (most recent call last):
12  File "<console>", line 1, in <module>
13  File "<console>", line 12, in max_pair
14  ValueError: Pas de valeur paire
```

2. Écrivez une fonction `afficher_max_pair`(liste) qui, en utilisant la fonction précédente, affiche un message donnant le plus grand nombre pair s'il existe ou le premier quatrain du poème « Art poétique » si la liste ne contient que des impaires.

```

1 >>> afficher_max_pair([9, 17, 8, 11, 4])
2 Le plus grand pair est 8
3 >>> afficher_max_pair([9, 7, -21, 11]) # Art poétique (Verlaine)
4 De la musique avant toute chose,
5 Et pour cela préfère l'Impair
6 Plus vague et plus soluble dans l'air,
7 Sans rien en lui qui pèse ou qui pose.

```

Exercice 4 — Reconnaître une liste triée

1. Écrivez une fonction `est_triee`(L) qui prend en argument une liste d'entiers L et qui renvoie `True` si la liste est triée en ordre croissant. Par exemple, `est_triee([1,2,2])` renvoie `True` et `est_triee([1,5,2])` renvoie `False`.
2. Écrivez des tests avec `assert`.

Exercice 5 — Compactage

Dans cet exercice, vous pouvez utiliser la méthode `append`.

1. Écrivez une fonction `grouper`(L) qui prend en argument une liste L et qui renvoie la liste obtenue en remplaçant toute suite d'éléments consécutifs x, x, x, \dots, x par un seul élément x. Par exemple, `grouper([4,4,4,2,2,4])` renvoie `[4,2,4]`.
2. Écrivez une fonction `compacter`(L) qui prend en argument une liste L et qui renvoie la liste obtenue par groupage en indiquant de plus à l'aide d'un couple la taille de chaque groupe. Par exemple, `compacter([4,4,4,2,2,4])` renvoie `[(3,4),(2,2),(1,4)]`.

Tri par insertion

Le tri par insertion est un algorithme de tri qui insère un à un les éléments à trier dans une liste qui contient à la fin le résultat attendu. Par exemple, on aura

```

Étape 0 : triés : []         à trier : [5,8,7,1]
Étape 1 : triés : [5]        à trier : [8,7,1]
Étape 2 : triés : [5,8]      à trier : [7,1]
Étape 3 : triés : [5,7,8]    à trier : [1]
Étape 3 : triés : [1,5,7,8]  à trier : []

```

Exercice 6 — Tri par insertion

1. Écrivez une fonction `index_insertion`(L,n) qui prend en arguments une liste triée d'entiers L et un entier n et qui renvoie l'indice de la position à laquelle insérer n dans L afin de garder la liste triée. Par exemple, `index_insertion([1,5,6,10],2)` renvoie 1 car 2 est à l'indice 1 dans la liste `[1,2,5,6,10]`. De plus, si n est déjà dans la liste, on renverra le plus grand indice qui convient. Par exemple, `index_insertion([1,2,6,10],2)` renvoie 2.
2. Écrivez une fonction `insertion_triee`(L) qui prend en argument une liste d'entiers L et qui renvoie une nouvelle liste contenant les entiers de L en ordre croissant. Vous pourrez utiliser la méthode `insert` sur les listes.