Les boucles `for`

Exercices - niveau basique

Tri de plusieurs listes

Écrivez une fonction qui accepte en argument une liste de listes, et qui retourne la même liste mais avec toutes les sous-listes triées.

```
# pour la correction, et un exemple
from corrections.w2_for import exo_multi_tri


# voici un exemple de ce qui est attendu
exo_multi_tri.exemple()
```

Écrivez votre code ici

```
def multi_tri(listes):
    "<votre_code>"
```

Vous pouvez le tester ici

```
exo_multi_tri.correction(multi_tri)
```

Tri de plusieurs listes, dans des directions différentes

Modifiez votre code pour qu'il accepte cette fois deux arguments listes que l'on suppose de tailles égales.

Comme tout à l'heure le premier argument est une liste de listes à trier.

À présent le second argument est une liste (ou un tuple) de booléens, de même cardinal que le premier argument, et qui indiquent l'ordre dans lequel on veut trier la liste d'entrée de même rang. True signifie un tri descendant, False un tri ascendant

```
# pour la correction, et un exemple
from corrections.w2_for import exo_multi_tri_reverse


# Pour être un peu plus clair, voici à quoi on s'attend
exo_multi_tri_reverse.exemple()
```

```
def multi_tri_reverse(listes, reverses):
    "<votre_code>"


# pour vérifier votre code
exo_multi_tri_reverse.correction(multi_tri_reverse)
```

Exercices - niveau intermédiaire

Liste des racines p-ièmes de l'unité

Dans le notebook sur l'utilisation de python comme un calculette, nous avions écrit de manière un peu fastidieuse les racines 3-ièmes de l'unité grâce à la formule

$r_n = e^{2i\pi \frac{n}{3}},$ pour $n\in {0,1,2}$

On vous demande à présent d'écrire une fonction qui retourne la liste de ces valeurs. Cette fois on n'utilisera plus le nombre '3', mais on le passera en argument à la fonction comme le nombre `p` que l'on peut supposer $>=2$

```
# retourne la liste des racine p-ièmes de l'unité
def liste_racines(p):
    "<votre_code>"


# pour vérifier votre code
from corrections.w2_for import exo_liste_racines
exo_liste_racines.correction(liste_racines)
```

Produit scalaire

On veut écrire une fonction qui retourne le produit scalaire de deux vecteurs. Pour ceci on va matérialiser les deux vecteurs en entrée par deux listes que l'on suppose de même taille. Cela est tout à fait possible avec le bagage que nous avons appris jusqu'ici - bien que nous verrons plus tard d'autres techniques pour faire ceci de manière plus élégante.

On rappelle que le produit de X et Y vaut

$\sum_{i} X_i * Y_i$

On posera que le produit scalaire de deux listes vides vaut `0`.

Vous devez donc écrire

```
def produit_scalaire(X,Y):
    """retourne le produit scalaire de deux listes de même taille"""
    "<votre_code>"
```

```
# pour vérifier votre code
from corrections.w2_for import exo_produit_scalaire
exo_produit_scalaire.correction(produit_scalaire)
```