Quelques exercices pour pratiquer

N'oubliez pas de documenter les fonctions. C'est indispensable pour bien écrire du code et plus prosaiquement, cela rapporte des points...

Exercice 1 : le tri bulle

Le tri bulle est un algorithme de tri (par défaut, nous considérons que le tri est fait dans l'ordre croissant). L'algorithme parcourt le tableau et compare les éléments consécutifs. Lorsque deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.

Après un premier parcours complet du tableau, le plus grand élément est forcément en fin de tableau, à sa position définitive. En effet, aussitôt que le plus grand élément est rencontré durant le parcours, il est mal trié par rapport à tous les éléments suivants, donc échangé à chaque fois jusqu'à la fin du parcours.

Après le premier parcours, le plus grand élément étant à sa position définitive, il n'a plus à être traité. Le reste du tableau est en revanche encore en désordre. Il faut donc le parcourir à nouveau, en s'arrêtant à l'avant-dernier élément. Après ce deuxième parcours, les deux plus grands éléments sont à leur position définitive. Il faut donc répéter les parcours du tableau, jusqu'à ce que les deux plus petits éléments soient placés à leur position définitive.

1. Proposez une fonction `rand` qui prend en argument un entier `N` et qui retourne une liste de taille `N` contenant des réels de l'intervalle $[0, 1]$ (on pourra utiliser la fonction `random` du module `random`.
2. Proposez une fonction `tri_bulle` qui prend en argument une liste et qui retourne la liste triée selon l'algorithme du tri par sélection.
3. Testez votre algorithme.
4. Modifiez votre fonction pour qu'elle accepte un argument optionnel qui sera une fonction d'ordre. Par défaut, vous prendrez la fonction `lambda x, y: True if x < y else False`.
5. Testez votre nouvelle version de l'algorithme.

In [34]:
```
from random import random

def rand(N):
    """retourne une liste de taille N contenant des réels alétoires entre 0 et 1"""
    l = []
    for _ in range(N):
        l.append(random())
    return l

def tri_bulle(T, f=lambda x, y: True if x < y else False):
    """tri bulle"""
    for j in range(len(T)):  # tri du j-eme élément en partant de la fin
        tbl_trie = True
        for i in range(len(T)-j-1):
            if not f(T[i], T[i+1]):
                T[i], T[i+1] = T[i+1], T[i]
                tbl_trie = False
        if tbl_trie:
            break
    return T
```

In [36]:
```
T = rand(5)
print(T)
print(tri_bulle(T))
print(tri_bulle(T, lambda x, y: True if x > y else False))
```

```
[0.3438359891975765, 0.6765102766689759, 0.6412514484226602, 0.7921010278329184, 0.18359260342241068]
[0.18359260342241068, 0.3438359891975765, 0.6412514484226602, 0.6765102766689759, 0.7921010278329184]
[0.7921010278329184, 0.6765102766689759, 0.6412514484226602, 0.3438359891975765, 0.18359260342241068]
```

Exercice 2 : construction d'une fonction qui affiche tous ses arguments...

Pour bien prendre en main la façon dont on accède aux arguments d'une fonction, nous allons fabriquer une fonction qui prend un nombre arbitraire d'arguments éventuellement rangés dans des listes ou des tuples et qui les affiche.

Question 1

- Proposez une fonction `affiche_0` qui prend un nombre arbitraire d'arguments et les affiche brutalement (sans se préoccuper du type des objets passés en argument).
- Documentez et testez votre fonction...

In [17]:
```
def affiche_0(*args):
    """affiche tous les arguments"""
    for argk in args:
        print(argk, end=' ')
```

In [18]:
```
affiche_0("toto", 1, [2, 3], [1, [2, 3]])
```

```
toto 1 [2, 3] [1, [2, 3]]
```

Question 2

- Modifiez la fonction précédente en une fonction `affiche_1` qui "entre" dans les listes, c'est-à-dire qu'elle affiche les éléments des listes plutôt que la liste complète.
- Documentez et testez votre fonction...

Indication : pour tester si un élément est une liste, vous pourrez utiliser la commande `isinstance` .

In [19]:
```
def affiche_1(*args):
    """affiche tous les arguments"""
    for argk in args:
        if isinstance(argk, list):
            for argki in argk:
                print(argki, end=' ')
        else:
            print(argk, end=' ')
```

In [20]:
```
affiche_1("toto", 1, [2, 3], [1, [2, 3]])
```

```
toto 1 2 3 1 [2, 3]
```

Question 3

- Modifiez la fonction précédente en une fonction `affiche_2` qui "entre" dans les listes jusqu'au plus bas niveau, c'est-à-dire qu'elle affiche les éléments des listes plutôt que la liste complète et cela même pour les listes de listes...
- Documentez et testez votre fonction...

Indication : pensez récursif...

In [23]:
```
def affiche_2(*args):
    """affiche tous les arguments"""
    for argk in args:
        if isinstance(argk, list):
            for argki in argk:
                affiche_2(argki)
        else:
            print(argk, end=' ')
```

In [24]:
```
affiche_2("toto", 1, [2, 3], [1, [2, 3]])
```

```
toto 1 2 3 1 2 3
```

Exercice 3 : manipulation de fonctions

- Créez une fonction `add` qui prend en argument deux fonctions f et g et qui retourne la fonction $f + g$.
- Créez une fonction `mul` qui prend en argument deux fonctions f et g et qui retourne la fonction $f * g$.
- Utilisez les deux fonctions précédentes pour fabriquer la fonction polynomiale $X^2 + 1$ et testez cette nouvelle fonction.

In [25]:
```
def add(f, g):
    """additionne deux fonctions"""
    def fpg(x):
        return f(x) + g(x)
    return fpg

def mul(f, g):
    """multiplie deux fonctions"""
    def fmg(x):
        return f(x) * g(x)
    return fmg
```

In [29]:
```
X = lambda x: x
Un = lambda x: 1

P = add(mul(X, X), Un)
print(P(2))
```

```
5
```

In [ ]:
```

```