# TP corrigé : listes 2
# Informatique pour tous

## 1 Fonctions de base

1. Écrire une fonction `somme` renvoyant la somme des éléments d'une liste
   ▶
```
def somme(L):
    res = 0
    for i in range(len(L)):
        res += L[i]
    return res
```
2. Écrire une fonction `positif` tel que `positif(L)` soit `True` si et seulement si tous les éléments de la liste `L` sont positifs.
   ▶ Si l'on trouve un nombre négatif, on peut renvoyer `False`. Si on a parcouru la liste en entier sans trouver de nombre négatif, on peut alors renvoyer `True` :
```
def positif(L):
    for i in range(len(L)):
        if L[i] < 0:
            return False
    return True
```
3. Écrire une fonction `minimum` renvoyant le minimum d'une liste non vide de nombres.
   Indice : on peut conserver dans une variable le minimum vu jusqu'à présent (qui contient initialement le premier élément de la liste). On parcourt ensuite la liste avec une boucle `for`, en mettant à jour le minimum à chaque fois qu'on trouve une valeur plus petite.
   ▶
```
def minimum(L):
    mini = L[0]
    for i in range(len(L)): # on peut commencer à 1...
        if L[i] < mini:
            mini = L[i]
    return mini
```

## 2 Opérations d'ensemble

1. Écrire une fonction `appartient(e, L)` pour savoir si un élément `e` appartient à une liste `L`.
   ▶ On parcourt `L`. Dès qu'on trouve `e`, on renvoie `True`. **Une fois parcouru la liste entière** sans avoir trouvé `e`, on renvoie `False`.
```
def appartient(e, L):
    for i in range(len(L)):
        if L[i] == e:
            return True
    return False
```
2. En déduire une fonction `inclus` telle que `inclus(L1, L2)` renvoie `True` si tous les éléments de `L1` appartiennent aussi à `L2`, `False` sinon.
   ▶ On parcourt `L1`. Si on trouve un élément `L1[i]` qui n'appartient pas à `L2` (en réutilisant la fonction précédente...), on peut renvoyer `False`. **Une fois parcouru la liste entière**, on peut renvoyer `True`.
```
def inclus(L1, L2):
    for i in range(len(L1)):
        if not appartient(L1[i], L2):
            return False
    return True
```

3. Écrire une fonction `intersection` telle que `intersection(L1, L2)` renvoie la liste composée des éléments à la fois dans `L1` et `L2`.
   ▶ On initialise une liste vide `res`. Puis on parcourt `L1` et on ajoute `L1[i]` à `res` si `L1[i]` appartient à `L2`.
```
def intersection(L1, L2):
    res = []
    for i in range(len(L1)):
        if appartient(L1[i], L2):
            res.append(L1[i])
    return res
```
4. Écrire une fonction `union` telle que `union(L1, L2)` renvoie la liste composée des éléments dans `L1` ou `L2`.
   ▶
```
def union(L1, L2):
    res = L1[:] # effectue une copie de L1
    for i in range(len(L2)):
        if appartient(L2[i], L1):
            res.append(L2[i])
    return res
```
5. Écrire une fonction `difference` telle que `difference(L1, L2)` renvoie la liste composée des éléments dans `L1` mais pas dans `L2`.
   ▶
```
def difference(L1, L2):
    res = []
    for i in range(len(L1)):
        if not appartient(L1[i], L2):
            res.append(L1[i])
    return res
```
6. En déduire une fonction `diff_symetrique` telle que `diff_symetrique(L1, L2)` renvoie la liste composée des éléments dans `L1` ou `L2`, mais pas les deux.
   ▶
```
def diff_symetrique(L1, L2):
    return difference(union(L1, L2), intersection(L1, L2))
```
7. Écrire une fonction `doublon` telle que `doublon(L)` détermine si la liste `L` contient un doublon (plusieurs fois le même élément).
   ▶
```
def doublon(L):
    for i in range(len(L)):
        if appartient(L[i], L[i+1:]):
            return True
    return False
```
   On rappelle que `L[i+1:]` est une suite extraite de `L` en en conservant que les éléments d'indices $\geq$ `i+1`.