TP 1

MANIPULATION DE LISTES

```
## Exercice 1

# 1

def disjoints(i1:[float], i2:[float]) -> bool:
    # if i1[1] < i2[0] or i2[1] < i1[0]:
    #     return True
    # else:
    #     return False
    return i1[1] < i2[0] or i2[1] < i1[0]


assert disjoints([1,2],[3,7])
assert not disjoints([1,8],[3,7])
assert not disjoints([1,8],[6,10])


# 2

def fusion(i1:[float], i2:[float]) -> [float]:
    if i1[0] < i2[0]:
        a = i1[0]
    else:
        a = i2[0]
    if i1[1] > i2[1]:
        b = i1[1]
    else:
        b = i2[1]
    return [a, b]


assert fusion([1, 2], [3, 7]) == [1, 7]
assert fusion([1, 8], [3, 7]) == [1, 8]
assert fusion([1, 3], [2, 5]) == [1, 5]


# 3

def verifie(L: [[float]]) -> bool:
    for i in range(len(L)-1):
        if L[i][1] >= L[i+1][0]:
            return False
    return True


assert not verifie([[0,1],[2,5],[3,6]])
assert not verifie([[2,5],[0,1],[3,6]])
assert verifie([[0,1],[2,3],[4,6]])
```

```python
def verifie_rec(L):
    if len(L) == 1:
        return True
    if L[0][1] >= L[1][0]:
        return False
    return verifie_rec(L[1:])


assert not verifie_rec([[0,1],[2,5],[3,6]])
assert not verifie_rec([[2,5],[0,1],[3,6]])
assert verifie_rec([[0,1],[2,3],[4,6]])


# 4

def appartient(x, L):
    for interv in L:
        if interv[0] <= x <= interv[1]:
            return True
    return False

def appartient_bis(x, L):
    for interv in L:
        if interv[0] > x:
            return False
        elif x <= interv[1]:
            return True
    return False


assert not appartient(-1,[[0,1],[2,3],[4,6]])
assert appartient(0,[[0,1],[2,3],[4,6]])
assert not appartient(3.5,[[0,1],[2,3],[4,6]])
assert appartient(5,[[0,1],[2,3],[4,6]])
assert appartient(6,[[0,1],[2,3],[4,6]])
assert not appartient(7,[[0,1],[2,3],[4,6]])




## Exercice 2

# 1

def rep(C):
    for i in range(len(C)):
        for j in range(i+1, len(C)):
            if C[i] == C[j]:
                return i
    return -1

# E = [[0 ,0] , [0 ,1] , [1 ,1] , [1 ,0] , [2 ,0] , [2 ,1] , [1 ,1] , [1 ,2] , [1 ,1] , [0 ,1]
     , [ -1 ,1]]


# 2

def boucle(C):
    d = rep(C)
    if d == -1:
        return []
    f = len(C)-1
    while C[f] != C[d]:
        f = f - 1
    return [d, f]
    # alternative au lieu du while:
    # for f in range(len(C)-1, d, -1):
    #     if C[f] == C[d]:
    #         return [d, f]
```

```python
# 3

def coupe(C):
    B = boucle(C)
    if B == []:
        return C
    d, f = B   # d, f = B[0], B[1]
    L = [C[i] for i in range(d+1)]
    for i in range(f+1, len(C)):
        L.append(C[i])
    return L



## Exercice 3

# Question 1

def estCroissante(L):
    if L == []:
        return True
    for i in range(len(L)-1):
        if L[i] > L[i+1]:
            return False
    return True


assert estCroissante([0,1,1,1,3,7])
assert not estCroissante([0,1,0,1,3,7])


def estDecroissante(L):
    if L == []:
        return True
    for i in range(len(L)-1):
        if L[i] < L[i+1]:
            return False
    return True

assert estDecroissante([4,2,1])


def estMonotone(L):
    return estCroissante(L) or estDecroissante(L)


# Question 2

def maxCroissante(L):
    a, b = 0, 0 # indices correspondant à la longueur maximale
    for i in range(len(L)):
        j = i
        while j < len(L)-1 and L[j] <= L[j+1]:
            j += 1
        if j-i > b-a:
            a, b = i, j
    return L[a:b+1]


assert maxCroissante([0, 1, 1, 3, 7]) == [0, 1, 1, 3, 7]
assert maxCroissante([1, 1, 0, 1, 1, 1, 3, 7, 6]) == [0, 1, 1, 1, 3, 7]
assert maxCroissante(list(range(0,-4,-1))) == [0]


# Question 3a
# La liste [0, 1, 0, 1, 0] ne présente que des monotonies banales.
```

```python
# Question 3b

def cahots(L):
    if len(L) < 2 or L[0] == L[1]:
        return False
    croissante = L[0] < L[1]  # booléen indiquant comment "on part"
    eprec = L[1]
    for e in L[2:]:
        if eprec == e:
            return False
        if eprec < e :
            if croissante:
                return False
        else :
            if not croissante:
                return False
        croissante = not croissante
        eprec = e
    return True


assert cahots([0,1])
assert cahots([0,2,1,3,2,3,0,1,0])
assert not cahots([1,2,3,2,1])


# Question 3c

from random import randint
L = [randint(-10,10) for n in range(15)]
L.sort()
print("Liste tirée :",L)

newL = []
n = len(L)

if n%2 == 0 : # nombre pair de termes
    for i in range(n//2):
        newL.append(L[:n//2][i])
        newL.append(L[n//2:][i])
else: # nombre impair de termes
    for i in range((n-1)//2):
        newL.append(L[:(n-1)//2][i])
        newL.append(L[(n-1)//2:][i])
    newL.append(L[(n-1)//2])

print("Liste réordonnée :",newL)
print("cahots(liste) ->",cahots(newL))
```