Correction DS Informatique : mouvement brownien

Quitte à se répéter : il existe quantité de solutions à chaque questions, on en propose ici quelques-unes parmi d'autres. L'ordre des solutions proposées est arbitraire. Le critère qui prime est la lisibilité, pas la concision (mais compte tenu de l'intérêt de certains pour les solutions courtes, on en propose une en exemple pour les questions simples).

1. On somme les valeurs absolues des déplacements :

```
def Long(L) :
    acc = 0
    for d in L :
        acc = acc + abs(d)
    return acc
```

```
def Long(L) :
    acc = 0
    for i in range(len(L)) :
        acc = acc + abs(L[i])
    return acc
```

```
def Long(L) :
    return sum([abs(d) for d in L])
```

2. On compte le nombre de déplacements strictements positifs aux positions paires :

```
def Droite(L) :
    acc = 0
    for d in L[::2] :
        if d > 0 :
            acc = acc + 1
    return acc
```

```
def Droite(L) :
    acc = 0
    for i in range(0, len(L), 2) :
        if L[i] > 0 :
            acc = acc + 1
    return acc
```

```
def Droite(L) :
    return sum([1 for d in L[::2] if d>0])
```

3. On compte les 4 et les -4 :

```
def NbMax(L) :
    acc = 0
    for d in L :
        if abs(d) == 4 :
            acc = acc + 1
    return acc
```

```
def NbMax(L) :
    acc = 0
    for i in range(len(L)) :
        if abs(L[i]) == 4 :
            acc = acc + 1
    return acc
```

```
def NbMax(L) :
    return sum([1 for d in L if abs(d) == 4])
```

Ou (sans doute à éviter en concours) :

```
def NbMax(L) :
    return L.count(4) + L.count(-4)
```

4. On somme les déplacements aux positions paires pour X, aux positions impaires pour Y.

Note : le sujet mentionne « couple », mais on pouvait retourner une « liste à deux éléments » également (mentionné par oral), et c'est le choix fait ici.

```
def Fin(L) :
    X, Y = 0, 0
    for i in range(len(L)) :
        if i%2 == 0 :
            X = X + L[i]
        else :
            Y = Y + L[i]
    return [ X, Y ]
```

Pour retourer un couple, on aurait écrit `return X, Y` ou `return (X, Y)`.

```python
def Fin(L) :
    res = [ 0, 0 ]
    for i in range(len(L)) :
        res[i%2] = res[i%2] + L[i]
    return res
```

```python
def Fin(L) :
    X, Y = 0, 0
    for i, d in enumerate(L) :
        if i%2 == 0 :
            X = X + d
        else :
            Y = Y + d
    return [ X, Y ]
```

```python
def Fin(L) :
    res = [ 0, 0 ]
    for i, d in enumerate(L) :
        res[i%2] = res[i%2] + d
    return res
```

```python
def Fin(L) :
    return [ sum(L[::2]), sum(L[1::2]) ]
```

5. Revoir le TP Météo pour celui-ci !

Il s'agit d'incrémenter un compteur pour tous les déplacements nuls, et de le remettre à zéro pour tous les déplacements non nuls. Et de conserver la plus grande valeur qu'atteint le compteur :

```python
def MaxImm(L) :
    compte = 0
    maxi = 0
    for d in L :
        if d == 0 :
            compte += 1
            if compte > maxi :
                maxi = compte
        else :
            compte = 0
    return maxi
```

C'est une question un peu plus difficile que les quatre premières, et probablement que certaines des suivantes, mais c'est un mécanisme qu'il est bon de retenir.

```python
def MaxImm(L) :
    compte = 0
    maxi = 0
    for i in range(len(L)) :
        if L[i] == 0 :
            compte += 1
            if compte > maxi :
                maxi = compte
        else :
            compte = 0
    return maxi
```

6. On reconstruit toutes les positions, et on compte celles égales à `[0, 0]`.

```python
def NbOrig(L) :
    pos = [ 0, 0 ]
    compte = 0
    for i in range(len(L)) :
        pos[i%2] = pos[i%2] + L[i]
        if pos == [ 0, 0 ] :
            compte += 1
    return compte
```

7. Même chose, mais on calcule les distances à l'origine, et on conserve la plus grande :

```python
def DistMax(L) :
    pos = [ 0, 0 ]
    dmax = 0.0
    for i in range(len(L)) :
        pos[i%2] = pos[i%2] + L[i]
        dist = (pos[0]**2 + pos[1]**2)**0.5
        if dist > dmax :
            dmax = dist
    return dmax
```

8. Pour qu'il y ait un cycle, il faut et suffit que `L[i] + L[i-2] == 0` et `L[i-1] + L[i-3] == 0`, pour $3 \leqslant i \leqslant n-1$:

```python
def NbCycl(L) :
    compte = 0
    for i in range(3, len(L)) :
        if L[i]+L[i-2]==0 and L[i-1]+L[i-3]==0 :
            compte += 1
    return compte
```

9. On reconstruit la liste des positions, et on cherche les traversées :

```
def NbCross(L) :
    pos = [ 0, 0 ]
    compte = 0
    for i in range(len(L)) :
        if (pos[i%2] * (pos[i%2] + L[i]) < 0) and pos[(i+1)%2] != 0 :
            compte += 1
        pos[i%2] = pos[i%2] + L[i]
    return compte
```

10. Un peu plus dur... On conserve la liste des positions, et pour chaque position, on cherche si on est déjà passé par là (uniquement toutes les quatres positions, et avec une distance au moins celle du plus long cycle déjà trouvé) :

```
def LongCycl(L) :
    lpos = [ [ 0, 0 ] ]
    lmax = 0
    for i in range(len(L)) :
        lpos.append( lpos[-1].copy() )
        lpos[-1][i%2] = lpos[-1][i%2] + L[i]
        for j in range(i+1-lmax, -1, -4) :
            if lpos[j] == lpos[-1] :
                lmax = i+1-j
    return lmax
```

Note : il existe des solutions plus efficaces, notamment à base de dictionnaires, on se contente d'une solution relativement élémentaire ici.