16/20

1) 4/6
2) 5/6
3) 7/8

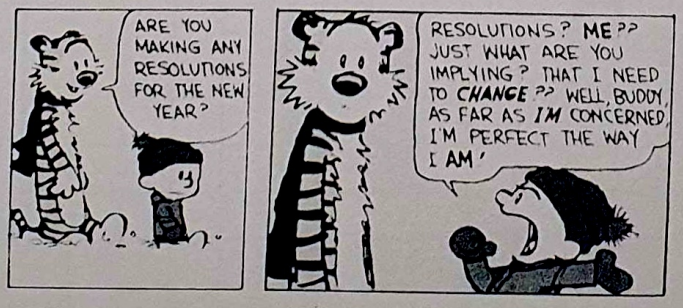
Algorithmique (Python)
Partiel n° 1 (B2)

INFO-SUP S1
EPITA

9 janvier 2024 - 9h45

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- **Le code :**
  - Tout code doit être écrit dans le langage `Python` (pas de C, CAML. ALGO ou autre).
  - **Tout code `Python` non indenté ne sera pas corrigé.**
  - Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
- Durée : 1h30



Exercice 1 (Liste somme cumulée – *6 points*)

Écrire une fonction `list_cumulated_sum(L, inf, sup)` avec :
- une liste d'entiers `L` supposée non vide
- deux entiers `inf` et `sup` supposés tels que $inf \leq sup$

qui construit et retourne une liste contenant chaque $i^{ème}$ élément de la liste `L` qui est tel que la somme des `i` premiers entiers de `L` est dans $[inf, sup]$.

Exemples d'applications :

```
>>> list_cumulated_sum([3, 5, 4, 2, 1], 3, 10)
[3, 5]
>>> list_cumulated_sum([7, 6, 5, 8], 5, 30)
[7, 6, 5, 8]
>>> list_cumulated_sum([1, 4, 2, 3], 15, 20)
[]
>>> list_cumulated_sum([30, 50, 20, 10, 40], 35, 70)
[30]
```

```
def list_cumulated_sum(L, inf, sup):
    resultList = []
    cumulSum = 0
    for i in range(len(L)):
        cumulSum += L[i]
        if cumulSum <= sup and cumulSum >= inf:
            resultList.append(L[:i])
    return resultList
```

optimisation à revoir

Exercice 2 (Monotone – *6 points*)

Une liste est considérée monotone si elle est soit entièrement croissante soit entièrement décroissante.

Écrire une fonction `monotonous(L)` qui vérifie si la liste d'entiers `L` est monotone.

Exemples d'applications :

```
>>> monotonous([])
True
>>> monotonous([1, 2, 3, 4, 5])
True
>>> monotonous([5, 5, 5, 5, 5])
True
>>> monotonous([5, 4, 3, 2, 1])
True
>>> monotonous([5, 5, 5, 2, 1])
True
>>> monotonous([1, 1, 2, 2, 2])
True
>>> monotonous([5, 5, 5, 6, 4, 3, 2, 1])
False
>>> monotonous([1, 2, 3, 2, 1])
False
```

```
def monotonous(L):
    croissant, decroissant = False, False
    if len(L) <= 2:
        return True
    i = 0
    while not(croissant and decroissant) and i < len(L)-2:
        if L[i] < L[i+1]:
            croissant = True
        elif L[i] > L[i+1]:
            decroissant = True
        i += 1
    return (i == len(L)-2) and not (croissant and decroissant)
```

solution pas parfaite avec traîne de tests

Exercice 3 (Majorité – *8 points*)

L'élément majoritaire d'une liste est un élément qui apparait plus de `n/2` fois dans la liste avec `n` la longueur de la liste.

1. Ecrire (ci-dessous) une fonction `my_max(L)` qui retourne la valeur maximum d'une liste `L` d'entiers naturels non vide.
2. Ecrire une fonction `majority(L)` (page suivante) avec :
   — une liste d'entiers naturels `L` supposée non vide
   qui retourne l'élément majoritaire de `L` s'il existe. `-1` sinon.

Exemples d'applications :

```
>>> majority([1, 2, 2, 2, 5, 2, 1])
2
>>> majority([3, 3, 4, 2, 4, 4, 2, 4, 4])
4
>>> majority([1, 2, 3, 4, 5])
-1
>>> majority([1, 2, 3, 3, 3, 3, 3, 4])
3
>>> majority([30, 30, 42, 22, 42, 42, 22, 42, 42])
42
>>> majority([10, 20, 30, 40, 50])
-1
>>> majority([10, 20, 10, 20])
-1
```

1. La fonction `my_max(L)` retourne la valeur maximum de la liste `L` d'entiers naturels non vide.

```
def my_max(L):
    max = L[0]
    for i in range(1, len(L)):
        if L[i] > max:
            max = L[i]
    return max
```

2. La fonction `majority(L)` retourne l'élément majoritaire de la liste d'entiers naturels L supposée non vide s'il existe, -1 sinon.

```
def majority (L):
    tempList = []
    for i in range(my_max(L)+1):
        tempList.append(0)
    for j in range (len(L)):
        tempList[L[j]] += 1
    Mmax = my_max (tempList)
    if Mmax < len(L)/2:
        return -1
    else:
        k = 0
        while tempList[k] != Mmax:
            k += 1
        return k
```

pas possible