1 Exercices préliminaires

Exercice 1. Niveau 1 : écrire une fonction de nom `deuxième` qui admet trois arguments entiers `a,b` et `c` et qui renvoie la valeur médiane. Exemple :

```
>>> deuxième(3, 6, 2)
    3

>>> deuxième (1, 5, 5)
    5
```

Indications : Bien sûr on peut tester les six configurations : a-t-on $a \leqslant b \leqslant c$ ou $a \leqslant c \leqslant b$ ou... (on s'arrête dès qu'on a trouvé la bonne bien sûr). Personnellement, je trouve cela assez lourd !

Deuxième méthode : on peut réordonner $a, b, c...$ (trier, donc) de sorte que $a \leqslant b \leqslant c$. On voit vite que pour y parvenir, il suffit de trois comparaisons, et, au maximum, trois échanges (pour lesquels, l'affectation simultanée est la plus adaptée : exemple `a, b = b, a`...)

Troisième méthode : déterminer au préalable le minimum et le maximum (les plus paresseux d'entre-vous feront appel à `min` et `max`)

Exercice 2. Niveau 1 : écrire une fonction de nom `somme` admettant un argument entier `n` et qui renvoie la somme $S_n = \sum_{i=1}^{n} \sum_{j=i}^{n} \frac{1}{1+i+j}$

Ecrire alors une fonction `premierIndice` qui admet un argument `v` (nombre entier ou flottant, peu importe) et qui renvoie le premier entier `n` tel que $S_n > v$. On fera bien sûr appel à la fonction précédente. Ici, on ne cherchera pas à être astucieux pour limiter le nombre d'appels à la fonction `somme`.

2 Arithmétique (un petit peu)

Exercice 3. Niveau 2 : écrire une fonction d'entête `convertit(n, b)` qui prend en paramètre un entier naturel `n` et une base `b` (élément de $\mathbb{N}^*$) et qui renvoie une liste formée des chiffres (présentés sous forme d'entiers) de l'écriture en base `b` de `n`. Par exemple :

```
>>> convertit(25, 13)
    [1, 12]
```

Modifier la fonction précédente pour qu'au lieu de renvoyer une liste, elle renvoie une chaîne de caractères. On supposera la base au plus égale à 36 et on écrira le chiffre 10 avec un A, 11 avec B et ainsi de suite jusqu'à, éventuellement, le chiffre 35 représenté par Z.

```
>>> convertit2(25, 13)
    "1C"
```

Indication : la fonction `chr` admet un argument entier, et renvoie le caractère associé (selon le codage unicode) à l'entier fourni en argument. La fonction `ord` réalise la fonction inverse : à un caractère, elle renvoie le code correspondant :

```
>>> ord("C")
    67
>>> chr(69)
    "E"
>>> ord("0")
    48
```

Sans surprise, les codes des lettres A,B,C,...,Z se suivent, comme se suivent les codes des caractères numériques 0,1,...,9.

Exercice 4. Niveau 2 : Ecrire la fonction réciproque d'entête `deBaseB(s, b)` qui admet deux arguments : une chaîne `s` qui représente un entier écrit selon la base `b` (entre 2 et 36) et qui renvoie l'entier ainsi représenté :

```
>>> deBaseB("1C", 13)
    25
```

3 Parcours de liste

Exercice 5. (fait en TP, mais à savoir refaire) Niveau 1 : écrire une fonction de nom `maximum`, admettant un unique argument de type liste (supposée non vide) et renvoyant la plus grande valeur de cette liste. Bien sûr, on s'interdira de faire appel à la fonction `max` de python !

Niveau 1.5 : reprendre la fonction précédente pour qu'elle renvoie non pas la plus grande valeur, mais l'indice de celle-ci. On pourra nommer `indiceMaximum` cette seconde fonction.

Exercice 6. Niveau 1.5 : écrire une fonction de nom `secondMaximum`, admettant un unique argument de type liste qu'on supposera comporter au moins deux éléments et qui renvoie la seconde plus grande valeur de cette liste. (Elle peut être égale à la plus grande, si cette plus grande valeur est présente deux fois).

Exercice 7. (fait en TP, mais à savoir refaire) Niveau 1 : écrire une fonction d'entête `indiceDe(L, v)` qui admet pour arguments une liste `L`, et une valeur `v`, et qui renvoie le premier indice où `v` figure dans `L` s'il y est présent, et `None` s'il n'en fait pas partie. (`None`, et non pas la chaîne `'None'` : ne soyez pas surpris en testant votre fonction que rien ne s'affiche dans la console, c'est bien là le comportement attendu.)

Exercice 8. (idem) Niveau 1,5 : écrire une fonction d'entête `indicesDe(L, v)` qui admet pour arguments une liste `L`, une valeur `v` et qui renvoie la liste (éventuellement vide) de tous les indices où `v` figure dans `L`. Par exemple :

```
>>> indicesDe([2,1,6,1,3,1], 1)
    [1,3,5]

>>> indicesDe([2, 7, 4], 3)
    []
```

Exercice 9. (déjà donné à la maison, mais peut-être ne l'avez-vous pas fait) Niveau 2 : écrire une fonction d'entête `sousListeCroissante(L)` qui admet pour argument une liste (qu'on supposera être de nombres, mais cela marcherait aussi pour une liste de chaînes, de tuples de nombres ou de tout type en python muni d'une relation d'ordre) et qui renvoie la plus grande longueur d'une sous-liste croissante (au sens large) formée de termes consécutifs de `L`. Par exemple :

```
>>> sousListeCroissante([1, 5, 6, 2, 5, 8, 1, 2, 0])
    3

>>> sousListeCroissante([4, 6, 1, 4, 6])
    3
```

puis reprendre la fonction précédente pour qu'elle renvoie non pas la longueur d'une plus grande sous-liste croissante, mais une telle sous-liste. Si plusieurs sous-listes sont possibles, on tâchera de renvoyer la première rencontrée (par un parcours de gauche à droite de notre liste bien entendu) :

```
>>> sousListeCroissante2([1, 5, 6, 2, 5, 8, 1, 2, 0])
    [1, 5, 6]

>>> sousListeCroissante([4, 6, 1, 4, 6])
    [1, 4, 6]
```

Exercice 10. Niveau 4 : écrire une fonction d'entête `sousListeCroissante(L)` qui admet pour argument une liste et qui renvoie la plus grande longueur d'une sous-liste croissante (au sens large) formée de termes non forcément consécutifs de `L` (mais tout de même d'indices croissants, il ne s'agit pas de trier en amont `L` pour donner comme réponse `len(L)`...). Par exemple :

```
>>> sousListeCroissante([1, 3, 2, 4, 3, 5])
    4
```

puis reprendre la fonction précédente pour qu'elle renvoie une telle sous-liste (d'indices dans `L` minimaux, ou n'importe laquelle après tout !) :

```
>>> sousListeCroissante2([1, 3, 2, 4, 3, 5])
    [1, 3, 4, 5]
```

Question subsidiaire : évaluer, selon la longueur `n` de `L`, la complexité de la votre fonction `sousListeCroissante`.