Les listes en Python

Le but de cette séance est de commencer à manipuler les listes en Python. Il est fortement conseillé de tester les éléments de syntaxe présentés dans le cours avant de vous lancer dans ce TD. Comme d'habitude, un rappel : la documentation Python est de très bonne qualité, utilisez-la (https://docs.python.org/3/tutorial/datastructures.html).

1 Travail à préparer chez vous avant la séance

1. Écrivez, en pseudo-code, un algorithme **impératif** calculant le terme d'indice n de la suite de Fibonacci définie par :

$$f_0 = f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}, \forall n \geq 2$$

2 Création de liste

La première chose à faire pour pouvoir manipuler des listes est d'apprendre comment les définir en Python. Il existe pour cela plusieurs syntaxes possibles (nous ne les verrons pas toutes aujourd'hui et reviendrons sur ce point lors d'un prochain TD). Le cas le plus simple est celui pour lequel la liste que l'on souhaite créer est de taille suffisamment réduite pour pouvoir l'écrire de manière explicite :

```
ma_liste = [1, 7, 9, 12]
```

Toutefois, dans certains cas, on souhaitera générer des listes très longues, par exemple, la liste des entiers de 0 à $n-1$ (avec n possiblement grand):

```
ma_liste = range(n)  # Entiers de 0 à n-1
ma_liste = range(n0, n)  # Entiers de n0 à n-1
```

2. Affichez la liste obtenue : que remarquez-vous ?

En fait, je vous ai menti, la fonction `range` ne retourne pas une liste mais un itérateur (`iterator`). Un itérateur est un flux de valeurs pour lequel la principale opération disponible est de demander la valeur suivante . Ainsi, si l'on souhaite juste parcourir de manière linéaire l'ensemble des entiers entre 0 et $n-1$, il n'y aura pas de différence entre l'utilisation d'une liste et celle d'un itérateur : la syntaxe Python sera la même et vous aurez même tendance à oublier que vous ne manipulez pas exactement une liste. Si toutefois vous deviez à tout prix manipuler une liste (mais cela est peu probable), il est possible de transformer un itérateur en liste :

```
ma_liste = list(range(n))
```

3 Comparaison d'approches récursive et impérative pour Fibonacci

3. Proposez une implémentation du pseudo-code réalisé à la première question de ce TD. Comparez le nombre d'opérations élémentaires de cette implémentation et de son pendant récursif. Pour cela, on pourra afficher un symbole `"."` sur la console à chaque itération :

   ```
   print(".", end="")
   ```

4 Recherche d'élément dans une liste

4. Écrivez une fonction qui prend en entrée une liste et un élément (par exemple un entier) et retourne l'indice de cet élément dans la liste (ou `None` si l'élément n'est pas présent dans la liste) sans faire appel à la méthode `index`.
5. Ecrivez une variante de la fonction précédente qui suppose que la liste est triée.
6. Écrivez une fonction qui prend en entrée une liste et un élément (par exemple un entier) et retourne une version de la liste passée en argument dans laquelle toutes les occurrences de cet élément ont été supprimées.

5 Exercice de synthèse

7. Écrivez une fonction qui prend un entier en entrée et retourne, s'il existe, l'indice n tel que f_n (le terme de rang n de la suite de Fibonacci) soit égal à cet entier (retourner `None` si un tel indice n'existe pas).