Exercice 1
Suite logique
On considère la suite logique suivante

1 11 21 1211 111221 ...

On se propose d'écrire un programme en `Python` qui détermine le n^e terme de cette suite (sous la forme d'une chaîne de caractère).

1. Écrire la fonction `convliste(L)` qui à partir d'une liste de la forme `L = [5, '1', 3, '2', 1, '3']` renvoie la chaîne de caractère `'513213'`.
2. Écrire une fonction `compte(i, s)` qui compte le nombre de caractères consécutifs identiques à `s[i]` à partir de l'indice `i`. `s` est une chaîne de caratère et `i` un indice valide. Cette fonction renvoie le nombre et le caractère `s[i]`.
3. Écrire la fonction `suitelogique(s)` qui donne le terme suivant la chaîne `s`.
4. Donner le 13^e terme de la suite logique du début de l'énoncé.
5. Tracer la courbe de la longueur du $n$ième terme en fonction de n pour $n \in [\![1, 25]\!]$.
6. Calculer les rapports de longueur successives des 25 premiers termes. Que conjecturer ?

Exercice 2
Algorithme de recuit simulé appliqué au problème du voyageur de commerce (☕)
On considère $A_0, \cdots, A_{n-1}$ n points du plan. On cherche un circuit passant par tous ces points en commençant par A_0 et en finissant par A_{n-1} et de longueur aussi petite que possible.
On représentera les abscisses et les ordonnées de ses points par deux listes python (que l'on notera `X` et `Y`).

1. Écrire une fonction `creepoints(n)` qui renvoie les listes `X` et `Y` correspondant à n points tirés au hasard dans $[\![-100, 100]\!]^2$.
2. Écrire une fonction `longueur(X, Y, sigma)` qui calcule la longueur de la ligne polygonale
$$\sum_{i=0}^{n-2} A_{\sigma(i)} A_{\sigma(i+1)}$$
où $\sigma(i)$ = `sigma[i]`, `sigma` étant une liste de n entiers parmi $[\![0, n-1]\!]$ correspondant à une **permutation** donnée des indices de $A_0, \cdots, A_{n-1}$. (c'est à dire que `sigma` est la liste des entiers $0, \ldots, n-1$ mais dans le désordre).
3. On ne connait d'algorithme général permettant de trouver le plus court chemin, c'est-à-dire la meilleure permutation σ' de $[\![1, n-2]\!]$ telle que
$$A_0, A_{\sigma'(1)}, A_{\sigma'(2)}, \cdots, A_{\sigma'(n-2)}, A_{n-1}$$
soit le plus court chemin de complexité inférieure à $(n-2)!$
Il existe un algorithme approché qui permet d'obtenir une solution proche de la solution optimale assez rapidement, on l'appelle **algorithme de recuit simulé**. Voici le principe.
On part de la permutation `sigma = [0,` $\underbrace{\texttt{1, 2, } \cdots \texttt{, n-2,}}_{\text{à permuter...}}$ `n-1]`$= \sigma$
On pose `T =` longueur du chemin $((A_i), \sigma)$ symbolisant la température
   - On choisit une permutation voisine. Pour cela, on pourra (pour simplifier) choisir i et j dans $[\![1, n-2]\!]$ et permuter i et j dans σ, on obtient σ'. On calcule $\delta =$ `longueur`$((A_i), \sigma')-$ `longueur`$((A_i), \sigma)$.
   (on pourra aussi transformer $i, i+1, \cdots, j$ en $j, j-1, \cdots, i+1, i$ et comparer les résultats)
   - si $\delta <= 0$, on remplace σ par σ'
   - sinon on remplace σ par σ' avec une probabilité de $e^{-\frac{\delta}{T}}$
   - on remplace T par $q \times$ T, q étant une constante proche de 1, < 1, typiquement $q = 0,99$.
   - on recommence cette recherche de permutation un nombre donné de fois, typiquement n^3.

   Écrire une fonction `recuit(X, Y, nb)` qui renvoie les listes `X1, Y1` correspondant au chemin obtenu après utilisation de l'algorithme de recuit simulé (on prendra $q = 0,99$ et une nombre d'itération de n^3).
   Représenter le chemin avant et après utilisation de cet algorithme.

INDICATION on pourra utiliser les fonctions python
```
from random import randint, random
from math import sqrt, exp
import matplotlib.pyplot as plt
```

Exercice 3
Sous-chaîne d'ADN (☕)
On se donne une chaîne de caractères parmi `'A'`, `'C'`, `'G'` et `'T'` représentant un brin d'ADN.
L'ADN se lit par mot de 3 lettres = un codon.
Le **codon d'initiation** est `TAC` (méthionine).
Les **codons stop** sont `ATT`, `ATC` et `ACT`.

1. Écrire une fonction `codon(ch)` où `ch` est une chaîne de trois caractères et qui renvoie `0` si la liste représente le codon d'initiation, `1` si la liste représente un codon stop et `-1` dans les autres cas.
2. Écrire une fonction `trouveADN(s)` qui en lisant de gauche à droite la chaîne `s` extrait le premier code encadré par la codon d'initiation et un codon stop. La fonction renvoie les indices du début des deux codons (`None` au lieu de l'indice si le codon n'a pas été trouvé).