Exercice 1
Triangle de Pascal (élémentaire)
Construire le tableau `numpy` suivant

$$\begin{pmatrix} 1 & & & & & \\ 1 & 1 & & & (0) & \\ 1 & & 1 & & & \\ 1 & & & 1 & & \\ 1 & & (0) & & 1 & \\ 1 & & & & & 1 \end{pmatrix}$$

puis le remplir pour obtenir le trangle de Pascal.

Exercice 2
Algorithme de Hörner (cours, élémentaire)
On cherche à calculer $\mathrm{P}(x)$ où $\mathrm{P} = a_{n-1}\mathrm{X}^{n-1} + \cdots + a_0$ en minimisant le nombre de multiplications. Voici un exemple

$$\begin{aligned}\mathrm{P} &= 4\mathrm{X}^3 - 7\mathrm{X}^2 + 3\mathrm{X} + 5 \\ &= 5 + \mathrm{X} \times (3 + \mathrm{X} \times (-7 + \mathrm{X} \times 4))\end{aligned}$$

Proposer une fonction `horner(P, x)` où P est donnée sous la forme d'une liste $[a_0, \cdots, a_{n-1}]$.

Exercice 3
Termes d'une suite (élémentaire)

1. Écrire une fonction `suiterec(u0, f, eps)` qui calcule les termes successifs de la suite
$$(u_n)_{n\in\mathbb{N}} : \begin{cases} u_0 = \texttt{u0} \\ u_{n+1} = f(u_n) \text{ pour } n \geqslant 0. \end{cases}$$
jusqu'à ce que $|u_{n+1} - u_n| <$`eps` puis renvoie le dernier terme obtenu ainsi que son indice.
2. Le tester avec u0 = 2, eps = 10^{-6} et $f(x) = \dfrac{1+x}{1+x^2}$ puis représenter la construction des termes de la suite sur une figure.

Exercice 4
Suites récurrentes croisées
On considère trois suites réelles (x_n), (y_n), (z_n) vérifiant

$$x_0 = y_0 = \frac{1}{2} \text{ et } z_0 = 0$$

1. On suppose que
$$\begin{cases} 10x_{n+1} = 7x_n + 4y_n + 5z_n \\ 10y_{n+1} = 3x_n + z_n \\ 10z_{n+1} = 6y_n + 4z_n \end{cases}$$
À l'aide d'un programme en `Python`, calculer $(x_{2015}, y_{2015}, z_{2015})$.
2. Quelle est la complexité en terme de nombre de multiplications de votre programme ? Pouvez-vous améliorer la complexité ?
3. On reprend le même problème (même condition initiale) mais avec le système
$$\begin{cases} x_{n+1} = 7x_n + 4y_n + 5z_n \\ y_{n+1} = 3x_n + z_n \\ z_{n+1} = 6y_n + 4z_n \end{cases}$$
Calculer $(x_{2015}, y_{2015}, z_{2015})$.

Exercice 5
Lecture de fichier (banque PT 2015)
On considère le fichier texte "`donnees.txt`" suivant

```
49987654
1, 3, 2015-08-31
2014-10-29, 08:34:15, 4568
2014-10-28, 20:21:48, 365
2014-10-28, 18:47:54, 987
```

1. Écrire un code en `python` permettant de lire ce fichier et de récupérer
en première ligne : le premier nombre (`id_titre`)
en 2e ligne : entier 1 (`zone1`), entier 2 (`zone2`), une liste [annee, mois, jour] (`date_fin`)
en 3e−suiv. lignes : une liste [annee, mois, jour] (`dates[i]`), une liste [annee, mois, jour] (`jours[i]`), un numero (`numeros[i]`)
2. Écrire une fonction `nbSecondesEntre(heure1, heure2)` prenant pour arguments deux horaires au format `[heures, minutes, secondes]` (donc sous forme de listes de trois entiers chacun) et retournant le nombre de secondes séparant les deux instants. Le résultat devra être positif si `heure1` est postérieure à `heure2`.

Exercice 6
Suite logique
On considère la suite logique suivante

$$1 \quad 11 \quad 21 \quad 1211 \quad 111221 \quad ...$$

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 13e terme de la suite logique du début de l'énoncé.

Exercice 7
Décomposition LU
On considère une matrice inversible $\mathrm{A} = (a_{i,j}) \in \mathfrak{M}_n(\mathbb{R})$.
On effectue des transformations sur la matrice A.
On notera les matrices $\mathrm{A}^{(0)} = 0$, $\mathrm{A}^{(1)}, \cdots, \mathrm{A}^{(k)} = (a_{i,j}^{(k)}), \cdots, \mathrm{A}^{(n-1)}$.
• Pour k variant de 1 à $n-1$,
pour $i \in [\![k+1, n]\!]$, on pose $\ell_{i,k} = \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}}$ (on suppose qu'à chaque étape $a_{k,k}^{(k-1)} \neq 0$)
et on effectue les transformations sur la matrice $\mathrm{A}^{(k-1)}$

$$\mathrm{L}_i \leftarrow \mathrm{L}_i - \ell_{i,k}\mathrm{L}_k$$

pour obtenir la matrice $\mathrm{A}^{(k)}$.
• On pose alors $\mathrm{U} = \mathrm{A}^{(n-1)}$ et $\mathrm{L} = (\widetilde{\ell}_{i,j})$ avec

$$\widetilde{\ell}_{i,j} = \begin{cases} \ell_{i,j} \text{ si } i > j \\ 1 \text{ si } i = j \\ 0 \text{ sinon} \end{cases}$$

On démontre que A = LU (on dit que l'on a effectué une décomposition LU de la matrice A).

1. Écrire une fonction `decompLU(A)` qui effectue cette décomposition et qui renvoie les matrices L et U.
2. Tester votre fonction sur la matrice $\mathrm{A} = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$
3. Comment résoudre simplement une équation du type AX = Y avec $\mathrm{X}, \mathrm{Y} \in (\mathfrak{M}_{n,1}(\mathbb{R}))^2$, X inconnu, lorsqu'on connaît une décomposition LU de la matrice A ?
4. Quelle est la complexité (en termes de multiplications) de la résolution de l'équation AX = Y par la décomposition LU, par la méthode de Gauss ?

Exercice 8
Fraction continue
On considère la fonction $f(x) = \dfrac{1}{x - \lfloor x \rfloor}$.
$\lfloor x \rfloor$=`floor(x)` en python (dans le module `math`).
Pour un nombre irrationnel $x_0 > 0$, on construit une suite $(u_n)_{n\in\mathbb{N}}$ telle que $u_0 = x_0$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = f(u_n)$.
On pose $a_n = \lfloor u_n \rfloor$. On dit que la suite (a_n) est le **développement en fraction continue** du nombre x_0.

1. Déteminer $a_0, \cdots, a_{10}$ pour $x_0 = e^1$ à l'aide de python.
2. On définit deux suites (p_n) et (q_n) de la manière suivante

$$p_0 = a_0,\ q_0 = 1,\ p_1 = a_0a_1 + 1,\ q_1 = a_1$$

et pour $n \geqslant 2$,

$$p_n = a_np_{n-1} + p_{n-2}$$
$$q_n = a_nq_{n-1} + q_{n-2}$$

et on pose $r_n = \dfrac{p_n}{q_n}$.
Calculer r_{10} avec l'exemple précédent et comparer r_{30} avec x_0.

Exercice 9
Qui gagne : le lièvre ou la tortue ?
Soit $p \geqslant 1$. Un lièvre et une tortue sont sur la ligne de départ. La tortue pour gagner doit avancer de p cases. Le lièvre doit juste se décider à partir et il a une chance sur 6 de le faire à chaque étape de la tortue.

On pourra utiliser le code suivant

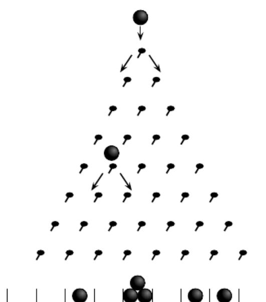
```
from random import randint
import matplotlib.pyplot as plt
from numpy import mean

def de6():
    return randint(1, 6)
```

1. Écrire une fonction `partie(p)` qui renvoie `True` si la tortue gagne (`False` sinon) dans une partie à `p` cases.
2. Écrire une fonction `frequence(N, p)` qui calcule la fréquence statistique où la tortue gagne, c'est-à-dire le nombre de parties où la tortue gagne divisé par le nombre total de parties.
3. Répresenter cette fréquence pour $p = 5$ avec `N` variant dans $\{100, 200, 300, \cdots, 2000\}$.
4. Pour limiter les écarts à la moyenne, on propose de prendre la moyenne de 10 fréquences statistiques pour chaque valeur de `N`. Représenter alors les moyennes obtenues. Vers quelle valeur semble-t-on converger ?
5. Quelle est la probabilité théorique que la tortue gagne ?

Exercice 10
La planche de Galton

On dispose de rangées de clous comme dans la figure ci-après. Soit N le nombre de clous de la dernière rangée. Sur la figure $N = 8$. On dispose de $N+1$ emplacements où viennent s'entasser des billes que l'on a lâché au sommet. Chacune de billes a une chance sur deux d'aller à gauche ou à droite. On numérote de gauche à droite chacun des emplacements de 0 à N. On cherche à connaître le nombre de billes dans chacun des emplacements au bout de n lancers.



1. Écrire une fonction `lancergalton(clous)` où `clous` vaut N le nombre de clous de la dernière rangée qui simule le lancer d'un bille et renvoie le numero de l'emplacement où atterrit la bille.
2. Écrire une fonction galton(clous, n) qui simule le lancé de n billes et renvoie une liste L où `L[k]` vaut le nombre de billes ayant atterri dans l'emplacement numéro $k \in [\![0,\texttt{clous}]\!]$. On pourra prendre `clous = 5`.
3. Afficher $\dfrac{\mathrm{L}[k] \times 2^{\texttt{clous}}}{n}$ pour $k \in [\![0,\texttt{clous}]\!]$. Que constate-t-on quand n est assez grand ? Proposer une explication.

Exercice 11
Autour des diviseurs d'un entier naturel $\geqslant 2$.

1. Écrire une fonction `diviseurs(n)` qui renvoie la liste des diviseurs de l'entier naturel n. On utilisera l'algorithme suivant :

   On départ la liste des diviseurs vaut `L = [1]`.

   On pose `d = 2`.

   Tant que $d \leqslant n$
   - on cherche les diviseurs d^k, $k \geqslant 0$ de n
   - on réactualise la liste `L`.
   - $n \leftarrow \dfrac{n}{d^p}$ avec d^p le plus grand diviseur trouvé précédemment
   - on incrément d.
2. Tester la fonction `diviseurs`, puis chercher les **nombres parfaits** dans $[\![2, 1000]\!]$, c'est-à-dire les nombres n qui sont égaux à la somme de leurs diviseurs (excepté n lui-même). Par exemple 6 ou 28 sont parfaits.
3. Soit $d_1, \cdots, d_p$ p nombres non nuls. Leur **moyenne harmonique** est le nombre m tel que
$$\frac{1}{m} = \frac{1}{p}\left(\frac{1}{d_1} + \frac{1}{d_2} + \cdots \frac{1}{d_p}\right)$$

   Écrire une fonction `moyharm(L)` qui renvoie la moyenne harmonique des nombres figurant dans le liste `L`.
4. Déterminer les **nombres à moyenne harmonique** dans $[\![2, 1000]$, c'est-à-dire les nombres dont la moyenne harmonique de ses diviseurs est un entier. On pourra pour cela écrire une fonction `testentier(a)` qui teste si le flottant a peut être considéré comme un entier (aux erreurs d'approximation près). (cette fonction n'est pas bien sûr pas 100% fiable).

5. Déterminer pour $n \in [\![2, 29]\!]$, les entiers n tel que $2^n\left(2^{n+1}-1\right)$ soit un entier à moyenne harmonique. Que remarque-t-on sur la moyenne harmonique ? Sur l'entier $2^{n+1}-1$?

Exercice 12
Tri à bulles
Le tri à bulle consiste à

- comparer deux nombres successifs du tableau et les réordonner, en commençant par les deux premiers, les deux suivants etc., jusqu'aux deux derniers. À cette étape, la « bulle » a fait remonter le plus grand élément en dernière position.
- On recommence (on compare `L[0]` et `L[1]` etc.) en s'arrêtant à l'avant dernier élément. Ainsi le 2e élément est mis en place
- On recommence jusqu'à un éventuel échange des deux premiers éléments.

Écrire une fonction `tribulle(L)` qui trie en place la liste `L`.
Quelle est la complexité (en nombre de comparaisons) ?

Exercice 13
Courbe du chien du jogueur
On considère un jogueur parcourant la courbe paramétrée suivante

$$\begin{cases} x(t) = (1+0,2\times\cos(5t))\cos(t) \\ y(t) = (1+0,2\times\cos(5t))\sin(t) \end{cases},\ t\in[0,15].$$

1. Tracer cette courbe en rouge.
2. À l'instant $t=0$, le chien du jogueur se trouve en $(0,10)$. Il court à une vitesse constante $\mathrm{V}=1$. Montrer que la courbe du chien $t\mapsto(\mathrm{X}(t),\mathrm{Y}(t))$ vérifie le système différentiel suivant

$$\begin{cases} \mathrm{X}'(t) = \mathrm{V}\dfrac{x(t)-\mathrm{X}(t)}{\sqrt{(x(t)-\mathrm{X}(t))^2+(y(t)-\mathrm{Y}(t))^2}} \\ \mathrm{X}'(t) = \mathrm{V}\dfrac{y(t)-\mathrm{Y}(t)}{\sqrt{(x(t)-\mathrm{X}(t))^2+(y(t)-\mathrm{Y}(t))^2}} \end{cases}$$

3. À l'aide d'un schéma d'Euler explicite (ordre 1), tracer la courbe du chien pour $t\in[0,15]$.
4. Vérifier en traçant la courbe du chien obtenue avec la fonction `odeint` du module `scipy.integrate`.

Exercice 14
Coïncidence lors d'un tirage complet
On dispose d'une urne avec n boules numérotées de 0 à $n-1$ et on effectue un tirage sans remise de ces n boules. On cherche à estimer la probabilité que le rang du tirage et le numéro de la boule coïncident au moins une fois.
On répère donc N fois le tirage complet et pour N grand, on peut penser que la fréquence statistique est proche de la probabilité théorique.
(c'est le théorème central limite, théorème hors-programme qui précise que N doit être choisi au moins de l'ordre de $\dfrac{1}{\varepsilon^2}$ si l'on veut une précision de ε).

1. Écrire une fonction `coincide(L)` qui à partir d'une permutation `L` des entiers dans $[\![0, n-1]\!]$ (donc `n = len(L)`) renvoie `True` ou `False` s'il existe $i\in[\![1,n]\!]$ tel que `L[i] = i`.
2. Écrire une fonction `simulation(n)` qui effectue une simulation avec n boules et renvoie `True` ou `False` suivant que la coïncidence est apparue ou non.

   Indication on pourra utiliser `del(L[i])` qui supprime le i^e élément de la liste `L`.
3. Écrire une fonction `freq(n, N)` qui effectue N tests statistiques avec n boules et renvoie la fréquence obtenue.

   La tester pour `n = 10` et `N = 10000`.

Rappel on utilisera la fonction `randint` du moudle `random` (interdiction d'utiliser les boites noires `shuffle` ou autres)

Exercice 15
Distance de Levenhstein (exemple de programmation dynamique)
On appelle distance de Levenshtein entre deux mots m1 et m2 le **nombre minimal** d'opérations pour aller de m1 à m2 parmi les opérations suivantes

- **substitution** d'un caractère de m1 en un caractère (différent) de m2
- **ajout** dans m1 d'un caractère de m2
- **suppression** d'un caractère de m1

On note `d[i, j]` le nombre minimal d'opérations pour passer du mot `m1[:i]` à `m2[:j]`.

1. Que vaut `d[0, j]` et `d[i, 0]` ?
2. Montrer que $d_{i,j} = \min\left(d_{i-1,j}+1,\ d_{i,j-1}+1,\ d_{i-1,j-1}+\delta^{\texttt{m2[j-1]}}_{\texttt{m1[i-1]}}\right)$.

   (δ est le symbole de Kronecker)

3. Écrire une fonction en `python` qui calcule la matrice $(d_{i,j})$ et qui renvoie le nombre minimal d'opérations pour passer du mot m1 au mot m2.
4. (☕☕) Montrer les différentes étapes de transformation pour passer du mot m1 au mot m2.
   INDICATION Faire un « backtracking » sur la matrice.

Exercice 16
Séparer des mots
On considère une chaîne de caractère `s`.
Écrire une fonction `separemot(s)` qui à partir de la chaîne de caractères `s` renvoie la liste des mots de la chaîne. Pour simplifier, on considèrera qu'un mot est constitué uniquement des lettres minuscules sans accent.
INDICATION on pourra utiliser la fonction suivante

```
def estlettre(c):
    return c >= 'a' and c <= 'z'
```

et rajouter une sentinelle à la chaîne de caractères `s, s = s + ' '`.

Exercice 17
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).

Exercice 18
Automate cellulaire : jeu de la vie de Conway
On se donne une matrice $[\![0, n+1]\!] \times [\![0, n+1]\!]$ dont les coefficients valent 0 (vide) ou 1 (présence d'une cellule vivante). Le bord de la matrice n'est constitué que de 0.
À chaque étape, on transforme la matrice suivant la règle suivante :

- chaque emplacement qui **n'est pas au bord** (pour simplifier) de la matrice possèdent 8 emplacements voisins.
  Pour un emplacement possèdant une cellule vivante (coefficient 1)
  - S'il possède 2 ou 3 voisins vivants, la cellule survit.
  - S'il possède 0, 1 ou au moins 4 voisins vivants, la cellule meurt.

  Pour un emplacement vide (coefficient 0)
  - S'il est entouré d'exactement 3 voisins vivants, il nait une nouvelle cellule.

1. Écrire une fonction `initialise(n, p)` qui renvoie une matrice (tableau `array`) avec une probabilité de présence p de cellule à l'intérieur (strict) de la matrice.
2. Écrire `vie(M)` qui renvoie la nouvelle matrice à l'étape suivante (on ne doit pas modifier l'ancienne matrice M)
3. Écrire une fonction `simulvie(n, p, N)` qui effectue une simulation du jeu de la vie avec N étapes et renvoie la matrice obtenue.
   Tester cette fonction. On pourra représenter graphiquement la matrice obtenue grâce au code suivant (si `M` est la matrice)

```
import matplotlib.pyplot as plt
import matplotlib.cm
def visualise(M):
    plt.figure('Jeu de la vie')
    plt.imshow(M, interpolation='nearest',
               cmap=matplotlib.cm.binary)
    plt.show()
```

   On pourra éventuellement utiliser `plt.pause(0.1)` pour obtenir une animation.

Exercice 19
Ruine d'un joueur
Pierre et Fabrice joue à pile ou face. La pièce a une probabilité p de donner pile.
Au départ, Pierre possède P_0 euros et Fabrice F_0.
Si la pièce donne pile, Fabrice donne un euro à Pierre (et l'inverse si la pièce donne face). Le jeu s'arrête quand l'un des deux joueurs est ruiné.
On considère la variable aléatoire P_k qui donne la fortune de Pierre à la k^e étape.
Bien sûr P_0 est constante et correspond à la fortune initiale de Pierre.
Le jeu s'arrête quand $\mathrm{P}_k = \mathrm{P}_0 + \mathrm{F}_0$ (Pierre gagne) ou quand $\mathrm{P}_k = 0$ (Pierre est ruiné).
On veut calculer les probabilités $a_{i,j} = \mathbb{P}(\mathrm{P}_i = j)$ pour $i \geqslant 0$ et $j \in [\![0, \mathrm{P}_0 + \mathrm{F}_0]\!]$.

1. Montrer que pour $i \geqslant 1$ et $j \in [\![1, \mathrm{P}_0 + \mathrm{F}_0 - 1]\!]$,
$$a_{i,j} = p \times a_{i-1,j-1} + (1-p) \times a_{i-1,j+1}$$
2. Donner une formule pour $a_{i,0}$, $a_{i,1}$, $a_{i,\mathrm{P}_0+\mathrm{F}_0-1}$ et $a_{i,\mathrm{P}_0+\mathrm{F}_0}$
3. Écrire une fonction `ruine(p, p0, f0, imax)` qui génère une matrice $(a_{i,j})$ (en bornant i par `imax`) sous la forme d'un `array` de `numpy`.
4. Représenter la loi de probabilité $(p_{\mathtt{imax},j})_{j \in [\![0,\mathtt{p0+f0}]\!]}$ pour un i fixé. On pourra utiliser la fonction `bar` de `matplotlib.pyplot`.

Exercice 20
Algorithme de recuit simulé appliqué au problème du voyageur de commerce (☕)
On considère $\mathrm{A}_0, \cdots, \mathrm{A}_{n-1}$ n points du plan. On cherche un circuit passant par tous ces points en commençant par A_0 et en finissant par A_n 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
$$\sum_{k=0}^{n-2} \mathrm{A}_{\sigma(i)}\mathrm{A}_{\sigma(i+1)}$$
où $\sigma(i) =$ `sigma[i]`, `sigma` étant une liste de n entiers parmi $[\![0, n-1]\!]$ correspondant à une **permutation** donnée des indices de $\mathrm{A}_0, \cdots, \mathrm{A}_{n-1}$.
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
$$\mathrm{A}_0, \mathrm{A}_{\sigma'(1)}, \mathrm{A}_{\sigma'(2)}, \cdots, \mathrm{A}_{\sigma'(n-2)}, \mathrm{A}_{n-1}$$
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,` $\underbrace{\mathtt{1,\ 2,}\ \cdots\mathtt{,\ n}-\mathtt{2,}}_{\text{à permuter...}}$ `n-1]` $= \sigma$
On pose `T =` longueur du chemin $((\mathrm{A}_i), \sigma)$ 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`$((\mathrm{A}_i), \sigma')-$ `longueur`$((\mathrm{A}_i), \sigma)$.
 (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 $e^{-\frac{\delta}{\mathrm{T}}}$
 - on remplace T par $q \times \mathrm{T}$, 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 21
Algorithme probabiliste (☕☕)
On considère le programme suivant

```
from random import random


def combalea(n, p, L):
    if n <= 0 or p <= 0 or n < p:
        return L

    r = random()
    if r < p/n:
        L.append(n)
        return combalea(n-1, p-1, L)
    else:
        return combalea(n-1, p, L)
```

1. Expliquer ce qu'affiche la suite d'instructions suivantes
`for i in range(20):`

```
L = []
print(combalea(27, 5, L))
```

2. On rappelle que $\dfrac{\binom{n-1}{p-1}}{\binom{n}{p}} = \dfrac{p}{n}$. Montrer que les sorties de la fonction `combalea` sont équiprobables.