Exercice 1
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 2^e ligne : entier 1 (`zone1`), entier 2 (`zone2`), une liste [annee, mois, jour] (`date_fin`)
   en 3^e-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 2
Triangle de Pascal (élémentaire)
Construire le tableau `numpy` (ou la liste de liste) suivant

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

puis le remplir pour obtenir le triangle de Pascal.

Exercice 3
On souhaite vérifier qu'une loi de Poisson peut être utilisée pour approximer une loi binomiale de paramètres n, p en posant $\lambda = n \times p$ le paramètre de la loi de Poisson.
On rappelle qu'une variable aléatoire X suit une loi binomiale de paramètres $n \in \mathbb{N}^*$ et $p \in]0,1[$ ssi

$$\forall k \in [\![0, n]\!] \mathbb{P}(\mathrm{X} = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

et qu'une variable aléatoire suit une loi de Poison de paramètres $\lambda > 0$ ssi

$$\forall k \in \mathbb{N}\ \mathbb{P}(\mathrm{X} = k) = e^{-\lambda} \frac{\lambda^k}{k!}$$

1. Créer deux versions de la fonction `factorielle(n)` qui calcule $n!$: une itérative et une récursive.
2. Implémenter la fonction `binom(n, k)` qui calcule et retourne $\binom{n}{k}$. La fonction devra retourner un résultat cohérent lorsque $k > n$.
   De plus, on limitera à 1 appel de la fonction factorielle et on prendra garde à calculer celui de $\binom{n}{k}$ et $\binom{n}{n-k}$ qui est le plus économe.
3. Créer les fonctions `loi_binom(n, p, max_k)` et `loi_poisson(lamb, max_k)` qui retournent les valeurs de $\mathbb{P}(\mathrm{X} = k)$ pour des valeurs de k allant de 0 à `max_k` (inclus).
4. Utiliser la fonction `plt.bar(X, Y, width=w, align='edge')` pour représenter sur un même graphique les lois binomiale ($n = 10, p = 0.2$ puis $n = 40, p = 0.1$) et de Poisson correspondante (cf. préambule) pour des valeurs de k allant de 0 à 10.
   On utilisera un paramètre `w` égal à 0.4 pour la loi binomiale et égal à -0.4 pour la loi de Poisson. Utiliser la commande help en cas de besoin.

Exercice 4
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 5
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.

Exercice 6
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 7
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 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

$$\sum_{k=0}^{n-2} A_{\sigma(i)} A_{\sigma(i+1)}$$

où $\sigma(i) =$ `sigma[i]`, `sigma` étant une liste de n entiers parmi $[\![0, n-1]\!]$ correspondant à une **permutation** donnée des indices de $A_0, \cdots, A_{n-1}$.

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 $e^{-\frac{\delta}{\mathrm{T}}}$
- 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
```