Tous les exercices sont précédés de la consigne ci-après : *Cet exercice est prévu pour le langage Python (et ses bibliothèques numpy, scipy, matplotlib). À chaque question, les instructions ou les fonctions écrites devront être testées.*

Exercice 1 Un nombre palindrome est un entier positif symétrique écrit dans une certaine base a comme ceci :

$$a_1 a_2 a_3 \cdots\cdots a_3 a_2 a_1.$$

Par exemple, il existe 90 nombres palindromes de trois chiffres :

$$\begin{gathered} 101,\ 111,\ 121,\ 131,\ 141,\ 151,\ 161,\ 171,\ 181,\ 191, \\ \cdots \\ 909,\ 919,\ 929,\ 939,\ 949,\ 959,\ 969,\ 979,\ 989,\ 999. \end{gathered}$$

Dans tout l'exercice, on manipulera des nombres écrits en base 10.

1. Écrire une fonction `Retourner` d'argument un nombre entier et renvoie son "*retourné*". On pourra utiliser en *Python* les instructions `str` et `int`.
2. Écrire une fonction `EstPalindrome` d'argument un nombre entier n et qui renvoie un booléen indiquant si n est un palindrome ou pas. La tester sur différentes valeurs.
3. Écrire une fonction `Palindromes` d'argument un nombre entier N qui renvoie la liste de tous les entiers palindromes inférieurs à N.

Nous avons le procédé suivant pour construire des nombres palindromes :

- Prendre un nombre au hasard (par exemple 39) ;
- lui ajouter le nombre "*retourné*" $(39 + 93 \to 132)$;
- à ce résultat, ajouter son nombre "*retourné*" $(132 + 231 \to 363)$;
- on répète le procédé jusqu'à obtention d'un nombre palindrome.

Attention : dans de rares cas ce calcul peut ne pas aboutir !

4. Écrire une fonction `ConstruitPalindrome` d'argument un nombre entier n et qui renvoie le nombre palindrome construit à partir de l'algorithme précédent.
5. Modifier cette fonction pour qu'elle renvoie aussi le nombre d'étapes nécessaires pour obtenir un palindrome.
6. Modifier cette fonction pour qu'elle s'arrête automatiquement au bout d'un nombre K de pas si un palindrome n'a pas été obtenu. Essayer de construire un nombre palindrome à partir de 196.

Exercice 2 On considère la suite (u_n) à valeurs réelles vérifiant :

$$u_0 = x,\ \ u_1 = y,\ \ u_2 = z,\ \ et,\ \ \forall n \geqslant 2,\ \ u_{n+3} = 2u_{n+2} + u_{n+1} - u_n.$$

On note λ, μ et ν les trois racines du polynôme $P = X^3 - 2X^2 - X + 1$ vérifiant $|\lambda| > |\mu| > |\nu|$.
On admettra dans le cadre de cet exercice qu'il existe pour toutes valeurs initiales x, y et z, trois coefficients réels a, b et c tels que :

$$\forall n \in \mathbb{N} \quad u_n = a\lambda^n + b\mu^n + c\nu^n$$

1. Écrire une fonction `LU` de quatre arguments N, x, y et z, qui renvoie la liste des N premières valeurs de la suite u_n.
   Faire afficher les 10 premières valeurs de la suite pour $x = 1$, $y = 2$ et $z = 3$.

2. À l'aide du quotient $\frac{u_{n+1}}{u_n}$, terme général d'une suite dont on déterminera la limite au brouillon, trouver une valeur approchée de λ qui rend la valeur du polynôme P inférieure à 10^{-10} en valeur absolue. On admettra que pour $x = 1$, $y = 2$ et $z = 3$, le coefficient a est non nul.

3. Trouver de même une valeur approchée de ν à l'aide du polynôme $Q = X^3P(1/X)$ et de la suite récurrente v_n associée.

4. En déduire une valeur approchée de μ.

Exercice 3 On définit la suite $(t_n)_{n\in\mathbb{N}}$ à valeurs dans $\{0, 1\}$ par :

$$t_0 = 0 \text{ et, } \forall n \in \mathbb{N}, \begin{cases} t_{2n} &= t_n \\ t_{2n+1} &= 1 - t_n \end{cases}.$$

1. Écrire une fonction récursive `t` d'argument un entier naturel n et renvoyant t_n.

On appelle "*mot T*" le mot binaire infini obtenu en concaténant les t_i :

`"0110100110010110 · · · "`

2. Écrire une fonction `mot` d'argument un entier naturel n non nul et renvoyant, sous forme d'une chaîne de caractères, le mot binaire constitué des n premiers caractères du *mot T*.
   Par exemple, l'appel de `mot(5)` doit renvoyer la chaîne `"01101"`.
   Faire afficher `mot(20)`.

3. Écrire une fonction `nbseq` de deux arguments, un entier naturel n non nul et une chaîne de caractères `seq`, calculant le nombre d'occurrences du mot binaire `seq` dans le mot `mot(n)`.
   Par exemple, l'appel `nbseq(15,"11")` doit renvoyer `3` car la séquence `"11"` apparaît 3 fois dans `mot(15)="0110100110010110"`. Attention, on comptera deux fois `"11"` dans `"111"`.

4. Conjecturer la limite de `nbseq(n,seq)/n` pour n tendant vers l'infini, pour tous les mots binaires `seq` de un, deux puis trois chiffres.

5. Quelle est la complexité de la fonction `mot` ? Quelle(s) amélioration(s) pourrait-on essayer d'apporter ?

Exercice 4 On considère $n + 1$ cases numérotées de 0 à n ; on place une bille dans la première case puis on répète n fois l'action suivante : faire avancer la bille d'une case ou la laisser dans sa case, avec équiprobabilité. Le résultat final de l'expérience est le numéro de la case dans laquelle se trouve la bille à la fin.

1. Créer une fonction `tirer` d'argument n qui effectue la simulation de l'expérience précédente et renvoie le numéro de la case dans laquelle se trouve la bille à la fin. On pourra utiliser la fonction `randint` du module `random`.

2. On effectue N fois le processus précédent. Pour $n = 10$ et $N = 500$, tracer les points représentant, en fonction du numéro k de la case, la proportion de billes se trouvant à la fin dans la case k.

3. Programmer une fonction renvoyant le coefficient binomial $\binom{n}{k}$.

4. Rajouter sur la figure précédente les points représentant la loi de probabilité associée à l'expérience.

5. Refaire ce qui précède en supposant que, maintenant, la bille a une probabilité p d'avancer d'une case (et $1-p$ de rester sur place). On pourra utiliser la fonction `random` du module `random`, qui tire une valeur entre 0 et 1 avec une distribution uniforme.

Exercice 5 Une fourmi ivre suit les lignes d'un quadrillage dont les intersections sont représentées par les couples de nombres entiers. Lorsqu'elle arrive à une intersection, elle a autant de chances de continuer tout droit, que de tourner à droite, de tourner à gauche, ou de revenir sur ses pas. On note C_a le carré formé des couples $[x, y]$ avec $-a \leqslant x \leqslant a$ et $-a\ y \leqslant a$.

1. Écrire une fonction `avancer` sans argument qui renvoie l'un des 4 couples `[1,0]`, `[0,-1]`, `[0,1]` ou `[-1,0]` avec équiprobabilité.
   On pourra utiliser la fonction `randint` du module `random`.

2. Simuler une trajectoire de la fourmi en partant de $[0, 0]$ puis en faisant afficher les noeuds du quadrillage par où elle passe, jusqu'à ce qu'elle sorte du carré C_2.

3. Écrire une fonction `traj` d'argument a qui renvoie une trajectoire tirée au hasard, partant du centre $[0, 0]$ et s'arrêtant lorsque la fourmi tente de sortir du carré C_a.

4. Tracer sur un même graphique 6 trajectoires différentes pour $a = 10$.

5. Soit L_a la longueur du trajet effectué par une fourmi pour sortir du carré C_a en partant de son centre. Définir une fonction `LM` de deux arguments a et N qui renvoie la moyenne des longueurs de trajet de N fourmis (N réalisations de la variable aléatoire L_a).

6. Tracer pour a entier compris entre 1 et 20, une estimation de $\mathbb{E}(L_a)$, l'espérance de L_a.

Exercice 6

1. Écrire une fonction `P0` d'argument un entier naturel N non nul renvoyant un vecteur de dimension $(N + 1)$ comportant un 1 en première composante et des zéros ailleurs.

2. Écrire une fonction T d'argument un entier naturel N non nul renvoyant la matrice carrée d'ordre $(N + 1)$ de terme général $(t_{tij})_{0\leqslant i,jjN}$ tel que :

$$t_{ij} = \begin{cases} 1 - j/N & \text{si } j = i - 1 \\ j/N & \text{si } j = i + 1 \\ 0 & \text{sinon} \end{cases}.$$

   On pourra utiliser la fonction `zeros` du module `numpy` de *Python*.
   Par exemple, `T(4)` doit donner la matrice $\begin{pmatrix} 0 & 0.25 & 0 & 0 & 0 \\ 1 & 0 & 0.5 & 0 & 0 \\ 0 & 0.75 & 0 & 0.75 & 0 \\ 0 & 0 & 0.5 & 0 & 1 \\ 0 & 0 & 0 & 0.25 & 0 \end{pmatrix}$.

3. Vérifier que la somme des coefficients de chaque colonne de la matrice `T(10)` vaut 1.

4. Pour une valeur de N donnée, on considère la suite récurrente de vecteurs $(P_n)_{n\in\mathbb{N}}$ définie par :

$$P_0 = \texttt{P0(N)} \quad \text{et} \quad P_{n+1} = \texttt{T(N)}\,P_n.$$

   Pour $N = 100$, faire tracer sur un même graphique les points représentant $P_n = [p_{0n}, p_{1n}, \cdots, p_{Nn}]$ (avec i en abscisses et p_{in} en ordonnées), pour $n = 20$, 50, 100 et 200.

5. Les i et les p_{in} décrivent en fait la loi de probabilité d'une variable aléatoire finie X_n :

$$\forall i \in \mathbb{N}, \quad 0 \leqslant i \leqslant N, \quad \mathbb{P}(X_n = i) = p_{in}.$$

   Définir une fonction `EX` de deux arguments N et n qui renvoie $\mathbb{E}(X_n)$, l'espérance de X_n.

6. Pour $N = 100$, faire tracer $\mathbb{E}(X_n)$ en fonction de n, pour $0 \leqslant n \leqslant 300$.
   Rajouter ensuite la courbe de $n \mapsto (N/2)(1 - \exp(-2n/N))$.

Exercice 7 Pour tout entier naturel n, on considère la fonction f_n définie sur $\mathbb{R}$ par : $f_n(x) = x^{n+1} - x_n - 1$.

1. Au brouillon, étudier les variations de f_n. Montrer notamment que f_n est strictement croissante sur $[1, +\infty[$ et que $f_n(2) > 0$ si $n \geqslant 1$.

2. On note α_n l'unique solution de $f_n(x) = 0$ sur $[1, 2]$. Que vaut α_0 ?

3. Tracer les courbes de f_n, pour n entier de 1 à 10. La plage d'affichage sera le rectangle $[1, 2] \times [-1, 1]$ (fonctions `xlim` et `ylim` du module `matplotlib.pyplot` en *Python*).
   Conjecturer le comportement de la suite $(\alpha_n)_{n\geqslant 0}$.

On rappelle ci-dessous l'algorithme de dichotomie.
Soit f continue sur un segment $[a, b]$ à valeurs réelles. On suppose que f s'annule exactement une fois sur $[a, b]$ en un point que l'on note α. On définit les suites $(a_k)_{k\geqslant 0}$ et $(b_k)_{k\geqslant 0}$ de la façon suivante :

- $a_0 = a$ et $b_0 = b$.
- On pose : $\forall k \in \mathbb{N}, \quad c_k = \dfrac{a_k + b_k}{2}$ et

$$\begin{aligned} \text{si } f(a_k)f(c_k) \leqslant 0, \text{ alors } a_{k+1} = a_k \text{ et } b_{k+1} = c_k, \\ \text{sinon } a_{k+1} = c_k \text{ et } b_{k+1} = b_k. \end{aligned}$$

On sait qu'alors les deux suites $(a_k)_{k\geqslant 0}$ et $(b_k)_{k\geqslant 0}$ convergent toutes les deux vers α, en vérifiant :

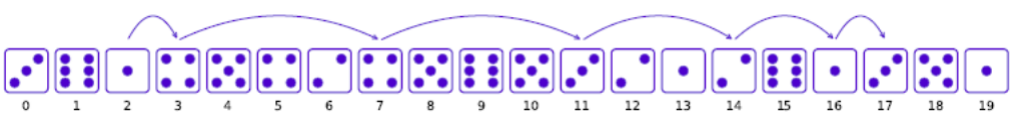
$$\forall k \in \mathbb{N}, \quad a_k \leqslant \alpha \leqslant b_k \text{ et } \forall k \in \mathbb{N}, \quad b_k - a_k = \frac{b-a}{2^k}.$$

On peut alors montrer que si $\dfrac{b-a}{2^k} \leqslant \varepsilon$, alors a_k et b_k sont des valeurs approchées de α à ε près.

4. En utilisant l'algorithme précédent, déterminer des valeurs approchées de α_n à 10^{-6} près pour n variant de 2 à 200. Quelle conjecture pouvez-vous faire ?

5. Pour ces mêmes valeurs de n, comparer graphiquement α_n avec la formule empirique $(1 + 1.6(n + 2.6)^{-0.83})$.

Exercice 8 On lance n dés cubiques usuels aux faces numérotées de 1 à 6. Les dés sont placés sur une ligne côte à côte. On choisit un dé dans le début de la liste et on avance, de la gauche vers la droite, d'autant de dés que la valeur marquée sur le dé départ. On recommence jusqu'à ce que cela ne soit plus possible. La situation est illustrée par le dessin ci-dessous.



1. Écrire une fonction `lancer` sans argument qui simule un lancer de dé et renvoie donc un entier entre 1 et 6. On pourra utiliser la fonction `randint` du module `numpy.random` de *Python*.

2. Écrire une fonction `liste` d'argument un entier n qui renvoie la liste des résultats de n lancers de dés.

3. Écrire une fonction `arrivee` de deux arguments, un entier k et une liste L de résultats, qui renvoie l'indice dans la liste L du dernier dé atteint en partant du dé numéro k, avec le procédé décrit dans l'énoncé.
   On testera la procédure en calculant les numéros d'arrivée pour toutes les positions de départ d'un tirage de 15, puis 20, puis 25 dés. Que constatez-vous ?

4. Écrire une fonction `commun`, d'argument une liste de dés L, qui renvoie le plus grand entier k tel que les dés en position $0, 1, \ldots, k$ aboutissent à la même position d'arrivée.

5. En utilisant la fonction `commun`, estimer la probabilité que les k premières positions d'une liste de n dés tirée au hasard aboutissent à la même position d'arrivée.

6. En déduire une estimation du nombre minimal n tel que les six premiers dés aient 95% de chances d'aboutir à la même position d'arrivée.