Cahier de TP d'informatique

2026-2027

Nom ..... Prénom .....

Table des matières

Thème 1 : Bases

I Range

■ **1.** *Nombre de termes d'une séquence `range`*

Sans utiliser la machine, indiquer le nombre de termes contenus dans les séquences suivantes :

a) `range(0,10)`.

b) `range(25)`.

c) `range(1,11)`.

d) `range(1,12,-1)`.

e) `range(12,2,-1)`.

■ **2.**

Générer les séquences de nombres suivantes à l'aide de `range` (on pourra vérifier que la solution proposée est correcte en programmant une boucle affichant les termes successifs du range)

a) 0,1,2,...,10

b) 1,2,3,...,14

c) 38,39,40,...,57

d) La séquence des 57 entiers consécutifs à partir de 41 (inclus).

e) La séquence des 33 entiers consécutifs à partir de −21 (inclus).

f) La séquence des 73 entiers consécutifs jusqu'à 84 inclus.

g) La séquence des 11 entiers consécutifs jusqu'à 104 exclu.

h) La séquence des entiers pairs de 0 à 44.

i) La séquence des entiers impairs de 0 à 44.

j) 12,15,18,..., 93.

k) 12,19,26,...180,187,194

l) 18,17,...,1

m) 18,16,...,2

n) 18,15,12,...,6

II Programmation basique de fonctions

■ **3.**

Programmer les fonctions suivantes :

a) $f_1 : x \mapsto x^2 - 2x - 1$.

b) $f_2 : x \mapsto \begin{cases} x^2+1 & \text{si} \quad x < 0 \\ x^3 & \text{si} \quad x \in [0,2] \\ \ln(x) & \text{sinon.} \end{cases}$

c) $f_3 : x \mapsto \dfrac{1}{1+\lfloor x^2 \rfloor}$

Rappel : $\lfloor t \rfloor$ désigne la partie entière de t. Elle s'obtient par la commande `floor` issue soit du module `math`, soit du module `numpy`.

d) $f_4 : x \mapsto \sqrt{x}\ln x$

III Boucles

■ **4.** *Choix du schéma itératif adéquat*

Dans chaque cas, dire si la programmation du schéma itératif demandée est plus adaptée avec **a)** une boucle `while`, **b)** une boucle `for` avec interruption de boucle, ou **c)** une boucle `for` sans interruption et dire pourquoi.

a) Le calcul des 30 premiers termes d'une suite donnée.

b) La simulation de 10 lancers d'une pièce de monnaie.

c) La recherche du premier terme d'une suite égal à 10.

d) La recherche du premier terme négatif ou nul d'une liste.

e) La recherche du premier caractère égal à `w` d'une chaine de caractères donnée.

f) La simulation d'un tirage successif avec remise d'une boule dans une urne jusqu'à l'obtention d'une boule blanche.

g) Le calcul du plus grand terme d'une liste.

■ 5.

Écrire en python la condition de la boucle `while` dans chacun des cas suivants :

a) Les termes consécutifs de la suite (u_n) sont calculés et stockés dans une variable `u`. On cherche le premier terme u_n pour lequel $u_n > 100$.

b) Les termes consécutifs de la suite (u_n) sont calculés et stockés dans une variable `u`. On cherche le premier terme u_n pour lequel $u_n > 100$ parmi $u_1, \dots, u_{50}$.

c) Les termes consécutifs de la suite (u_n) sont calculés et stockés dans une variable `u`. On cherche le premier terme u_n de rang $n > 1000$ pour lequel $u_n > 100$.

d) On simule une suite de tirages successifs avec remise de boules numérotées. Les numéros des deux derniers tirages successifs sont stockés dans l'ordre dans des variables `s` et `t`. On s'arrête quand la suite des numéros tirés n'est plus croissante.

e) On simule une suite de tirages successifs avec remise de boules numérotées. Les numéros des deux derniers tirages successifs sont stockés dans l'ordre dans des variables `x`,`y`,`z`. On s'arrête quand la suite des numéros tirés n'est plus monotone.

IV Calcul de sommes

■ 6.

À l'aide d'une boucle `for`, programmer dans chaque cas une fonction prenant en entrée un entier `n`, et renvoyant sortie la valeur de S_n dans les cas suivants :

a) $S_n = \sum_{k=1}^{n} k$.

b) $S_n = \sum_{k=0}^{n} 2^k$.

c) $S_n = \sum_{k=0}^{2n} \frac{k(k-1)}{2}$.

d) $S_n = \sum_{k=0}^{n} u_k$ où $u_n = \frac{1}{n}$ si n est impair, et $u_n = e^{-n}$ sinon.

V Suites

■ 7. *Suite explicite*

Écrire une fonction `u(n)` prenant en entrée un entier `n` et renvoyant en sortie la valeur du terme u_n de la suite définie par :

$$u_1 = 3 \quad \forall n \geq 1 \quad u_{n+1} = \frac{nu_n - 1}{2}$$

Si vous ne vous êtes pas trompés, vous devez obtenir u_{15} =`-1035072.125`.

■ 8.

Suite récurrente Écrire une fonction `liste(n)` prenant en entrée un entier n et renvoyant en sortie la liste des termes $[u_0, \dots, u_n]$ de la suite récurrente définie par :

$$u_0 = 0 \quad \forall n \geq 0 \quad u_{n+1} = \ln(2 + u_n)$$

. On construira la liste :

a) Par concaténations successives dans une première version de la fonction.

b) Par usage de `append` dans une seconde version.

■ 9. *Suite récurrente à deux pas*

Écrire une fonction `Fibo(n)` qui prend en entrée un entier n et qui renvoie en sortie le terme F_n de la suite de Fibonacci. Pour rappel, la suite de Fibonacci est définie par :

$$\begin{cases} F_0 = F_1 = 1 \\ \forall n \geq 0 \quad F_{n+2} = F_{n+1} + F_n \end{cases}$$

■ 10. *Série harmonique*

On pose $H_n = \sum_{k=1}^{n} \frac{1}{k}$. Écrire une fonction `rang(M)` prenant en entrée un flottant M et qui renvoie en sortie le plus petit entier n tel que $H_n > M$. Si vous

ne vous êtes pas trompés, `rang(10)` devrait renvoyer `12367`.

VI Listes

VI – A Opérations sur les listes - Slicing

■ 11.

Recopier le script suivant :

```
n = 15
L = [k**2 for k in range(1, n+1)]
```

a) Extraire de la liste `L` par slicing les sous-listes suivantes :

1) La sous-liste `L1` des 3 premiers termes de `L`.

2) La sous-liste `L2` des 5 derniers termes de `L`.

3) La sous-liste `L3` contenant les termes d'indice impair de `L`.

4) La sous-liste `L4` contenant les termes d'indice pair de `L`.

5) La sous-liste `L5` contenant un terme sur trois de `L` en partant du premier.

6) La sous-liste `L6` contenant un terme sur cinq de `L` en partant du deuxième.

b) Construire la liste `M` obtenue en insérant dans `L` un `0` en indice `8` (par slicing et concaténation).

c) Construire la liste `R` des termes de `L` rangés en sens inverse.

■ 12.

a) Écrire une fonction `reverse(L)` prenant en entrée une liste `L` et renvoyant en sortie la liste dans laquelle les items de `L` sont rangés en sens inverse (par boucle).

b) Même question en utilisant le slicing.

■ 13.

Écrire une fonction `saute(L,p)` qui prend en entrée une liste `L`, une entier `p` et qui renvoie en sortie la sous-liste constitué d'un terme sur deux de `L` à partir de son terme d'indice `p`.

VI – B Listes en compréhension

■ 14.

Pour tout entier strictement positif, on note :

$$a_n = \frac{1}{n} \quad \text{et} \quad b_n = 2^n$$

Construire les listes suivantes en compréhension contenant les mêmes objets que les ensembles mathématiques suivants (les p-listes mathématiques seront représentées en python par des listes ici, et non pas des tuples) :

a) $E_1 = \{a_k \quad k \in \{1, \dots, 15\}\}$.

b) $E_2 = \{b_k \quad k \in \{1, \dots, 15\}\}$.

c) $E_3 = \left\{ \frac{1}{n(n+1)} \quad n \in \{1, \dots, 10\} \right\}$.

d) $E_4 = \{|\cos n - \sin n| \quad n \in \{1, \dots, 10\}\}$

e) $E_5 = E_1 \setminus \{1/3, 1/11, 1/8\}$.

f) $E_6 = \{(a_k, b_k) \quad k \in \{1, \dots, 15\}\}$. *Le couple (a_i, b_i) sera représenté par un tuple en python.*

g) $E_7 = \{(2,4,8,16), (3,9,27,81), (4,16,64,256), (5,25,125,625)\}$.

h) $E_8 = \{(1), (2,3), (3,4,5), (4,5,6,7), (5,6,7,8,9)\}$. *Les p-listes seront représentées par des tuples en python.*

i) $E_9 = \{(i,j) \quad i \in \{-1,0,1\}, j \in \{-1,0,1\}\}$ *Les couple seront représentés par des listes en python.*

j) $E_{10} = E_9 \setminus \{(0,0)\}$.

VI – C Exercices complémentaires sur les listes

■ 15.

Écrire une fonction `diff(L)` prenant en entrée une liste de flottants `L` et renvoyant en sortie la liste des `L[i+1]-L[i]`.

■ 16.

Écrire une fonction `cumsum(L)` prenant en entrée une liste de flottants `L` et renvoyant en sortie la liste `S` dont le k-ème item vaut $\sum_{i=0}^{k} \texttt{L[i]}$.

■ 17.

Écrire une fonction `est_dans(L1,L2)` prenant en entrée deux listes `L1` et `L2` et renvoyant en sortie `True` si tous les éléments de `L2` sont dans `L1` et `False` sinon.

■ 18.

Écrire une fonction `maxi(L)` prenant en entrée une liste de flottants et renvoyant en sortie la valeur de son plus grand élément.

■ 19.

Écrire une fonction `pos_max(L)` prenant en entrée une liste de flottants et renvoyant en sortie la liste des indices en lesquels se trouve son plus grand élément. Utiliser l'exercice précédent.

■ 20.

Écrire une fonction `sans_rep(L)` prenant en entrée une liste `L` et renvoyant en sortie la liste des valeurs distinctes de `L`. Par exemple, si `L=[1,3,'toto',0,2,1,2,'2','toto']`, la sortie est `[1,3,'toto',0,2,'2']`.

Indication : Partir d'une liste `M` initialement vide, parcourir `L` et ajouter l'élément courant dans `M` si il n'est pas déjà dans `M`. À la fin, `M` contient tous les objets de `L` sans répétition.

■ 21.

Écrire une fonction `multiplicite(L)` prenant un entrée une liste `L` et renvoyant en sortie le tuple `(L1,M)` où `L1` est la liste des items distincts de la liste `L` et `M` est la liste contenant en position `i` le nombre d'occurrence dans `L` de `L1[i]`. Par exemple, si `L=[1,3,'toto',0,2,1,2,'2','toto']`, la sortie est :

`([1,3,'toto',0,2,'2'],[2,1,2,1,2,1]).`

VII Tracés de courbes de fonctions

■ 22.

Tracer les courbes des fonctions suivantes en définissant :

— la liste des abscisses à l'aide de `linspace`.

— la liste des images par une liste en compréhension, ou en exploitant la vectorisation des fonctions du module `numpy` (si vous préférez).

a) $f(x) = x^2 - x + 1$ sur $[0,1]$ avec 100 points régulièrement espacés.

b) $f(x) = x\cos(2\pi x)$ sur $[-1,1]$ avec 100 points régulièrement espacés.

c) $f(x) = x^2 - 2$ sur $[1,2]$ avec 100 points régulièrement espacés.

d) $f(x) = \dfrac{1}{x^2+1}$ sur $[0,1]$ avec 100 points régulièrement espacés

e) $f(x) = xe^{-x}$ sur $[0,+\infty[$ avec 200 points régulièrement espacés

f) $f(x) = \ln(1+x)$ sur $[1,2]$ avec 100 points régulièrement espacés.

g) $f(x) = \arctan x$ sur $\mathbf{R}_+$ avec 100 points régulièrement espacés

h) $f(x) = \dfrac{\ln x}{x}$ sur $]0,1]$ avec 100 points régulièrement.

Thème 2 : Calcul intégral et équations différentielles

I Méthode des rectangles

■ 23.

Calculer une valeur approchée des intégrales suivantes par une somme de Riemann avec 100 rectangles :

a) $I_1 = \int_0^2 2 - x^2 \, \mathrm{d}x$

b) $I_2 = \int_1^2 \ln(x) - 1 \, \mathrm{d}x$

c) $I_3 = \int_1^6 \sqrt{x} - 2 \, \mathrm{d}x$

d) $I_4 = \int_0^3 x - 2\sin(2x) \, \mathrm{d}x$

e) $I_5 = \int_0^1 \frac{\mathrm{d}x}{1+x^2}$.

■ 24.

Reprendre l'exercice précédent en calculant encore l'intégrale par la méthode de rectangle mais cette fois avec un pas $h = 10^{-3}$. On utilisera la fonction `arange` du module `numpy`.

■ 25. *Suite définie par une intégrale*

On considère la suite $(u_n)_{n\geq 0}$ définie par :

$$\forall n \in \mathbf{N} \quad u_n = \int_0^1 x^2 \sin(nx) \mathrm{d}x.$$

a) Écrire une fonction `u(n)` qui prend en entrée un entier n et qui renvoie en sortie une valeur approchée de u_n calculée par la méthode des rectangles avec 100 rectangles.

b) Écrire une fonction `liste(n)` qui prend en entrée un entier n et qui renvoie en sortie la liste des termes $u_0, \ldots, u_n$ calculée précédemment.

c) Conjecturer la nature de la suite (u_n).

II Fonctions définies par une intégrale

■ 26.

Tracer les courbes des fonctions suivantes sur les intervalles I donnés :

a) $F_1(x) = \int_0^x e^{-t^2} \mathrm{d}t \quad I = [0,1]$

b) $F_2(x) = \int_0^x \frac{\mathrm{d}t}{1+t^2+t^4} \quad I = [-5,5]$

■ 27.

a) Tracer sur l'intervalle $[0,10]$ la courbe de la fonction

$$f : x \mapsto \int_x^{x^2} \sin(4t) \mathrm{d}t$$

b) Tracer également sur un même graphique la solution exacte que vous aurez calculée au préalable.

c) Commenter et expliquer.

■ 28. *Intégrale à paramètre*

Tracer chacune des fonctions suivantes sur les intervalles donnés :

a) $F_1(x) = \int_0^1 \frac{e^{-tx}}{1+t} \mathrm{d}t$, $I = [1,5]$.

b) $F_2(x) = \int_0^1 \frac{\sin(x+t)}{x+t} \mathrm{d}t$, $I = [2,10]$.

c) $F_3(x) = \displaystyle\int_0^1 \frac{\exp\left(-x\left(1+t^2\right)\right)}{1+t^2}\mathrm{d}t$, $I = [-1,1]$.

■ **29.**

Tracer chacune des fonctions suivantes sur les intervalles donnés :

a) $F_1(x) = \displaystyle\int_0^1 \frac{e^{-tx}}{1+t}\mathrm{d}t$, $I = [1,5]$.

b) $F_2(x) = \displaystyle\int_0^1 \frac{\sin(x+t)}{x+t}\mathrm{d}t$, $I = [2,10]$.

c) $F_3(x) = \displaystyle\int_0^1 \frac{\exp\left(-x\left(1+t^2\right)\right)}{1+t^2}\mathrm{d}t$, $I = [-1,1]$.

III Méthode d'Euler

■ **30.** *Schéma d'Euler*

Écrire le schéma d'Euler pour chacune des équations différentielles suivantes pour un pas $h > 0$ donné et en précisant l'expression de la variable t_k en fonction de t_0.

a) $y' = y$, $\quad t \in [0,1]$, $t_0 = 0 \quad y(t_0) = 1$.

b) $y' - 2y = 0$, $t \in [0,1]$, $t_0 = 0 \quad y(t_0) = 1$.

c) $y' + y = t$, $t \in [0,1]$, $t_0 = 0 \quad y(t_0) = 1$.

d) $y' + ty = \dfrac{t}{2}e^{-\frac{t^2}{2}}$, $t \in [0,1]$, $t_0 = 0$, $y(t_0) = 1$.

e) $y' + 2y = z(t)$ (z : fonction donnée), $t \in [0,1]$, $t_0 = 0$, $y(t_0) = 1$.

f) $y' = F(t,y)$, (F : fonction donnée).

■ **31.** *Système différentielles d'équations couplées*

Même question que l'exercice précédent pour le système différentiel suivant :

$$\begin{cases} x' = -2x + y \\ y' = 2x - 2y \end{cases}$$

avec : $t_0 = 0, \quad t \in [t_0, 1], \quad (x(t_0), y(t_0)) = (1,0)$

■ **32.**

Tracer les solutions approchées des équations différentielles des exercices précédents sous python (on prendra le pas les valeurs $h = 10^{-1}$, $h = 10^{-2}$)

■ **33.**

a) On considère l'équation différentielle suivante

$$\begin{cases} y' + 2xy = xe^{-x^2}y^3 & \text{sur } I = \mathbf{R} \\ y(0) = 1 \end{cases}$$

Un calcul direct permet de constater que la fonction $x \mapsto \sqrt{\dfrac{3}{e^{-x^2} + 2e^{2x^2}}}$ est solution.

b) Calculer sur $[0,2]$ une solution approchée avec la méthode d'Euler et un pas égal à $h = 2 \cdot 10^{-2}$.

c) Tracer sur un même graphique les solutions exacte et approchée.

■ **34.**

a) On considère l'équation différentielle suivante

$$\begin{cases} y' + \dfrac{2}{x}y = x^2 & \text{sur } I = \mathbf{R}_+^\star \\ y(0) = \dfrac{2}{5} \end{cases}$$

Calculer la solution exacte de l'équation.

b) **1)** Calculer une solution approchée par la méthode d'Euler avec un pas $h = 0,025$ sur $[0,05;2]$.

2) Tracer sur un même graphique les solutions exacte et approchée.

c) Même question que la question précédente avec un pas égal à $h = 5$ sur l'intervalle $[1,200]$.

■ **35.**

On considère le système différentiel avec conditions initiales suivant :

$$\begin{cases} x' = -2x + \dfrac{1}{4}y \\ y' = -3x \\ (x(0), y(0)) = (1,-1) \end{cases}$$

a) Vérifier que le couple de (x,y) de fonctions données par :

$$x : t \mapsto -\frac{3}{4}e^{-t/2} + \frac{7}{4}e^{-3t/2} \quad y : t \mapsto -\frac{9}{2}e^{-t/2} + \frac{7}{2}e^{-3t/2}$$

est solution.

b) Calculer par la méthode d'Euler la solution approchée sur $[0,10]$ avec un pas égal à $h = 0,1$.

Thème 3 : Résolution approchée de l'équation $f(x) = 0$

I Méthode de dichotomie

■ **36.** *mathématiques élémentaires*

Si $a < b$ sont deux réels, donner l'expression m du milieu du segment $[a, b]$ et de sa longueur ℓ (vérifiez la pertinence de votre réponse pour $a = 1$, $b = 101$).

■ **37.**

Utiliser la méthode de dichotomie pour calculer toutes les solutions à $\varepsilon = 10^{-9}$ près des équations suivantes (on commencera par tracer un graphique pour trouver un intervalle sur lequel appliquer la méthode).

a) $x^5 - x^4 - 5x^3 - x^2 + 4x + 3 = 0$

b) $x^5 - x^4 - 5x^3 - x^2 + 4x + 3 = 2$

c) $x^2(4 - x^2) = \dfrac{4}{x^2 + 1}$.

d) $e^{-x} = 2 + x$.

e) $\ln(4 - x^2) = x$.

f) $\sqrt{x^2 - x + 1} = 2\sin(\pi x)$

■ **38.** *Méthode par balayage*

Soit $\varepsilon > 0$. La méthode par balayage consiste à calculer une valeur approchée $x_\star$ à ε près de l'équation $f(x) = 0$ pour $f \in \mathscr{C}^0([a, b])$ de la manière suivante :

a) On initialise $x_\star$ à $x_0 = a$.

b) On considère la suite (x_k) de terme général $x_k = x_0 + k\varepsilon$.

c) On compare les signes de $f(x_k)$ et $f(x_{k+1})$.

d) Dès que pour un entier k, ces nombres sont de signe contraire, on déduit que $x_k \leq x_\star \leq x_{k+1}$, ce qui fournit un encadrement d'amplitude ε de $x_\star$.

Reprendre l'exercice précédent en calculant cette fois une solution approchée par balayage.

II Suites implicites

■ **39.**

a) Écrire une fonction prenant en entrée un entier $n \geq 3$ et renvoyant en sortie la liste des termes de la suite $(u_n)_{n\geq 3}$ calculés à 10^{-4} près, où u_n est défini comme l'unique solution positive de l'équation

$$x^n + x^2 + 2x - 1 = 0.$$

Conjecturer la nature et la monotonie de la suite (u_n).

b) Écrire une fonction prenant en entrée un entier $n \geq 3$ et renvoyant en sortie la liste des termes de la suite $(u_n)_{n\geq 3}$ calculés à 10^{-4} près, où u_n est défini comme l'unique solution positive de l'équation

$$x^n(x - 1) = e^{-x}.$$

Conjecturer la nature et la monotonie de la suite (u_n).

On pourra commencer par tracer les courbes de fonctions pour déterminer un segment sur lequel chercher à calculer u_n.

■ **40.**

Améliorer les programmes de l'exercice précédent afin que le script fournisse aussi le nombre d'itérations réalisées pour atteindre la précision souhaitée.

III Méthode de Newton

■ 41.

Utiliser la méthode de Newton pour donner une valeur approchée à 10^{-9} près des nombres suivants :

a) $\sqrt[3]{30}$.

b) $\sqrt[7]{1000}$.

■ 42.

Utiliser la méthode de Newton pour donner une valeur approchée à 10^{-9} près de la solution positive à l'équation $2\sin x = x$.

■ 43.

Reprendre l'exercice 41 , en calculant les valeurs approchées par la méthode de dichotomie. Donner le nombre d'itérations nécessaires pour atteindre le résultat demandé.

IV Dichotomie sur un intervalle non borné

■ 44.

Soit $n \in \mathbf{N}^*$ et f_n la fonction définie sur $\mathbf{R}_+$ par

$$f_n : t \mapsto e^{-t}\left(1 + t + \frac{t^2}{2!} + \cdots + \frac{t^n}{n!}\right).$$

a) Programmer la fonction `f(n,t)` qui prend en entrée un entier `n`, un flottant `t`, et renvoie en sortie la valeur de $f_n(t)$.

b) Tracer la courbe de différentes fonctions f_n pour des valeurs de votre choix de n.

c) On s'intéresse à l'équation $f_n(t) = 1/2$. On peut montrer que l'équation admet une unique solution t_n.

Conjecturer graphiquement la nature de la suite (t_n) .

On souhaite programmer une fonction `solution(n,a,b)` qui prend entrée un entier `n` et renvoie en sortie une valeur approchée de t_n par dichotomie sur l'intervalle $[a,b]$

d) Programmer la fonction `solution`.

e) Pourquoi ne peut-on pas calculer une valeur approchée de t_n par dichotomie sur un intervalle $[a,b]$ fixé ?

f) Expliquer ce que fait le script suivant :

```
1 a,b,n=0,1,20
2 while (f(n,a)-0.5)*(f(n,b)
      -0.5)>0:
3     a,b=b,b+2*(b-a)
4 solution(n,a,b)
```

Thème 4 : Calcul matriciel

On pourra se reporter au formulaire de l'oral du concours Agro-Veto

```
1 import numpy as np
2 import numpy.linalg as la
```

I Construction de matrices et manipulation

■ 45.

```
1 # Exemple de saisie d'une
    matrice M en donnant la
    liste de ses lignes
2 M = np.array([[1,2,3],
3                [4,5,6]])
4 # Obtention de son format
5 M.shape
```

a) Créez une matrice A de taille $(3,4)$ avec des éléments de votre choix. Vérifiez ses format en utilisant la méthode `shape`.

b) Transformez la matrice A en une matrice de taille $(2,6)$ en utilisant la méthode `reshape`, puis en une matrice $(6,2)$. Quelle contrainte doit respecter le produit $n \times p$ pour effectuer cette opération ?

c) Créez une matrice B de taille $(5,2)$. Transformez-la en une matrice de taille $(2,5)$, puis en une matrice $(10,1)$. Affichez les format à chaque étape.

d) Créez une matrice uniligne (de taille $(1,5)$) et une matrice unicolonne (de taille $(5,1)$). Affichez leurs format et vérifiez que vous obtenez les formats attendus avec `shape`.

e) Créez une matrice C de taille $(6,3)$. Utilisez `reshape` pour la transformer en une matrice $(3,6)$, puis en une matrice $(2,9)$. Vérifiez les format avec `shape`.

II Construction de matrices avec des formules

■ 46.

a) Complétez, affichez A et corrigez si besoin.

```
1 # Construction de la matrice
    de taille 3x4 de coeffs
    a_ij = i+2j
2 A = np.zeros((3,4))
3 for i in range(3):
4     for j in range(4):
5         A[i,j] =
```

b) Créez une matrice B de taille $(3,3)$ telle que l'élément en position (i,j) soit $b_{i,j} = i + j$ $(1 \leq i, j \leq 3)$. Affichez cette matrice. Affichez cette matrice. Vérifiez les valeurs des éléments.

c) Construisez une matrice C de taille $(4,4)$ telle que l'élément en position (i,j) soit $c_{i,j} = i \times j$ $(1 \leq i, j \leq 4)$. Affichez cette matrice. Vérifiez les valeurs des éléments.

d) Créez une matrice D de taille $(3,4)$ telle que l'élément en position (i,j) soit donné par $d_{i,j} = i - j$. $(1 \leq i \leq 3,\ 1 \leq j \leq 4)$ Affichez cette matrice. Vérifiez les valeurs des éléments.

e) En utilisant des boucles `for`, générez une matrice E de taille $(5,5)$ telle que chaque élément soit défini par $e_{i,j} = i^2 + j^2$ $(1 \leq i, j \leq 5)$.

f) Construisez une matrice F de taille $(4,3)$ telle que chaque élément soit défini par $f_{i,j} =$

$(-1)^{i+j} \times (i+j)$. Affichez cette matrice et vérifiez sa construction.

g) Reprendre les exemples précédents en utilisant la commande `np.array` et en construisant la liste des lignes en compréhension.

III Extraction de coefficients, lignes et colonnes

■ 47.

a) Soit A une matrice de taille $(4,5)$ avec des éléments de votre choix. Écrivez le code pour extraire :

- le coefficient de la première ligne et deuxième colonne ;
- la première ligne de A ;
- la dernière colonne de A.

b) Affichez la sous-matrice formée par les lignes 1 à 3 et les colonnes 2 à 4.

c) Créez une matrice B de taille $(5,5)$. Extrayez les éléments diagonaux, puis tous les éléments situés au-dessus de la diagonale principale.

d) Soit C une matrice de taille $(6,4)$. Extrayez les lignes 2 et 4, puis les colonnes 1 et 3.

e) Construisez une matrice D de taille $(7,7)$. Affichez la sous-matrice formée par les lignes et colonnes impaires.

IV Matrices remarquables

■ 48.

a) Utilisez `np.zeros((n, p))` et `np.ones((n, p))` pour créer respectivement une matrice nulle et une matrice remplie de 1 et de taille $(3,3)$.

b) Créez la matrice identité de taille $(4,4)$ avec `np.eye`.

c) Créez une matrice E de taille $(5,5)$ ayant des 1 sur le contour et des 0 à l'intérieur, en utilisant des combinaisons de `zeros` et `ones`.

d) Construisez une matrice diagonale de taille $(4,4)$ avec des valeurs de 2 sur la diagonale en utilisant `np.eye`.

e) Créez une matrice F de taille $(6,6)$ avec des 1 sur la diagonale principale et -1 sur les premières diagonales (c'est-à-dire : au-dessus et en dessous de la diagonale principale).

V Produit matriciel

■ 49.

a) **1)** Si $A\,B$ sont deux matrices de formats compatibles, et $C = AB$, rappeler l'expression du coefficient général $c_{i,j}$ de la matrice C.

2) Implémentez une fonction `mult(A, B)` pour le produit de deux matrices A et B formats compatibles, sans utiliser les fonctions de produit de `numpy`.

b) Testez votre fonction avec deux matrices de format $(2,3)$ et $(3,2)$, puis vérifiez avec `np.dot`.

c) Multipliez la matrice A par elle-même pour calculer A^2, puis A^3. Comparez les résultats avec `np.dot`.

d) Créez deux matrices aléatoires de votre choix de format $(3,3)$. Calculez leur produit avec `mult` et `np.dot`.

e) Écrivez une fonction qui calcule le produit de trois matrices $(A \times B \times C)$ et vérifiez sa compatibilité pour des matrices de format $(2,2)$, $(2,3)$, $(3,2)$.

VI Puissances de matrices

■ 50.

a) **1)** Implémentez une fonction `puissance(A, n)` qui renvoie la puissance n-ième d'une matrice carrée A, en multipliant successivement la matrice par elle-même.

2) Implémentez une version récursive de votre fonction.

b) Comparez les résultats avec `la.matrix_power`.

c) Testez votre fonction sur une matrice identité de taille $(3,3)$ et vérifiez que $A^n = A$ pour $n > 1$.

d) Calculez la puissance 5 d'une matrice diagonale $(3,3)$ avec des valeurs de 2, 3, et 4 sur la diagonale.

e) Pour une matrice $B = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, calculez B^4 et comparez avec les résultats de `matrix_power`.

VII Utilisation du module `numpy.linalg`

■ **51.**

a) Calculez le rang des matrices suivantes :

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

b) Calculez l'inverse de $C = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -1 & 2 \\ 4 & 0 & -1 \end{pmatrix}$ avec `la.inv` et vérifiez que $C \times C^{-1} = I_3$.

c) Soit $E = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix}$. Calculez le rang, les valeurs propres, et essayez de déterminer si E est diagonalisable.

d) Calculez le rang d'une matrice nulle de taille $(4,4)$ et d'une matrice remplie de 1 de taille $(4,4)$.

Thème 5 : Simulation en calcul des probabilités

I Jeu de pile ou face

```
1 import random as rd
```

■ 52.

Objectif : Écrire une fonction Python simulant un lancer de pile ou face, puis l'utiliser pour estimer la probabilité d'obtenir face.

a) Écrire une fonction `pile_ou_face()` qui simule un lancer de pile ou face, en renvoyant en sortie équiprobablement `0` ou `1`. On pourra utiliser la commande `randint`.

b) Simuler 1000 lancers et calculer la fréquence d'apparition de face.

c) Augmenter le nombre de lancers jusqu'à 10 000 et observer la convergence de la fréquence vers la probabilité théorique.

II Estimation de probabilités par simulations répétées

■ 53.

Objectif : Estimer la probabilité d'un événement en répétant une simulation (repose sur la loi faible des grands nombres).

a) Écrire une fonction `tirage()` pour simuler un tirage aléatoire de boules rouges et bleues dans un sac (5 rouges, 3 bleues). La fonction renverra `1` pour une boule rouge et `0` pour une boule bleue. On pourra utiliser la fonction `choice` et construire une liste `U` représentant le sac.

b) Estimer la probabilité de tirer une boule rouge en simulant 1000 tirages.

c) Comparer la fréquence observée avec la probabilité théorique.

III Variables de Bernoulli

■ 54.

Objectif : Simuler une variable de Bernoulli pour observer la probabilité d'un succès.

a) Écrire une fonction simulant une variable de Bernoulli avec succès de probabilité $p = 0,3$. La fonction renverra 1 avec probabilité p (et 0 avec la probabilité $1-p$).

b) Répéter l'expérience 1000 fois et calculer la fréquence des succès.

c) Observer la fréquence des succès pour différentes valeurs de p.

IV Schéma de Bernoulli

■ 55.

Objectif : Étudier un schéma de Bernoulli avec plusieurs répétitions d'une expérience.

a) Simuler 10 répétitions d'une expérience de Bernoulli avec succès de probabilité $p = 0.2$ et déterminer le nombre de succès.

b) Répéter ce processus 1000 fois pour estimer la probabilité d'obtenir exactement 3 succès.

c) Comparer cette probabilité avec le résultat théorique.

V Simulation de tirages pour un jeu de dé

■ **56.**

Objectif : Étudier des tirages indépendants sur un dé équilibré.

a) Simuler 1000 lancers de dé et calculer la fréquence d'apparition de chaque face.

b) Comparer les fréquences obtenues avec les probabilités théoriques.

c) Estimer la probabilité d'obtenir deux 6 consécutifs dans une série de 1000 lancers.

VI Distribution de la somme de deux dés

■ **57.**

Objectif : Étudier la distribution de la somme obtenue lors du lancer de deux dés.

a) Simuler 1000 lancers de deux dés et calculer la somme des deux résultats.

b) Calculer les fréquences des sommes obtenues, et comparer avec la distribution théorique.

VII Variables géométriques : temps jusqu'au premier succès

■ **58.**

Objectif : Simuler une variable géométrique pour observer le temps d'attente jusqu'à un succès.

a) Écrire une fonction qui simule la répétition d'une épreuve de Bernoulli jusqu'au premier succès avec $p = 0.3$.

b) Simuler 1000 répétitions et calculer la moyenne du nombre de tentatives nécessaires.

c) Comparer la moyenne obtenue à l'espérance théorique $E(X) = \frac{1}{p}$.

VIII Temps d'apparition du r-ième succès

■ **59.**

Objectif : Étudier le temps d'apparition du r-ième succès dans une séquence de répétitions d'épreuves de Bernoulli.

a) Écrire une fonction qui simule le nombre de tentatives pour obtenir le 5ème succès avec $p = 0.3$.

b) Simuler 1000 répétitions et calculer la moyenne du nombre de tentatives nécessaires.

IX Temps d'apparition de la séquence pile-pile-face

■ **60.**

Objectif : Étudier le temps d'apparition d'une séquence donnée dans une répétition de lancers de pile ou face.

a) Écrire une fonction simulant des lancers de pile ou face jusqu'à obtenir la séquence pile-pile-face. Par exemple, La fonction devra renvoyer 6 si la série de lancer observée est `FPFPPF`.

b) Simuler 1000 répétitions et calculer la moyenne du nombre de lancers nécessaires pour obtenir cette séquence.

c) Comparer la moyenne observée à des valeurs théoriques connues.

X Longueur de la première série unicolore dans une urne

■ **61.**

Objectif : Observer la longueur de la première série unicolore dans une séquence de tirages.

a) Écrire une fonction simulant les tirages successifs avec remise d'une boule noire ou blanche dans une urne (contenant une boule noire et une boule blanche), et renvoyant en sortie. la longueur de la première série de boules de même couleur.

b) Répéter la simulation et calculer informatiquement la longueur moyenne de cette première série unicolore.

XI Temps d'attente jusqu'à l'obtention de deux succès consécutifs

■ 62.

Objectif : Étudier le temps d'attente pour obtenir deux succès consécutifs lors de répétitions mutuellement indépendantes d'épreuves de Bernoulli.

a) Écrire une fonction simulant des répétitions d'une expérience de Bernoulli avec $p = 0.5$.

b) Observer le nombre de tentatives nécessaires pour obtenir deux succès consécutifs.

c) Simuler 1000 répétitions pour calculer la moyenne de cette attente.

XII Simulation d'une marche aléatoire

■ 63.

Objectif : Simuler une marche aléatoire sur une droite.

a) Écrire une fonction qui simule une marche aléatoire d'un point sur une droite, avançant d'une unité à droite ou à gauche avec des probabilités égales.

b) Tracer l'évolution de la position du point au cours de 100 étapes.

c) Étudier les propriétés de cette marche en répétant l'expérience plusieurs fois.

Thème 6 : Fonctions polynomiales

Rappel : *L'ensemble* $\mathbf{R}_n[X]$ *désigne l'ensemble des polynômes de degré au plus* n*. C'est un espace vectoriel.*

a) On choisit de manipuler les polynômes en utilisant les *listes* comme structure de données pour représenter les coefficients.

b) Un polynôme P de degré *au plus* n est représenté par une liste de longueur $n+1$, où chaque élément de la liste correspond au coefficient du monôme de degré correspondant.

c) Par exemple, le polynôme $P = 3X^3+2X-5$ vu comme élément de $\mathbf{R}_3[X]$ sera représenté par la liste `[ -5, 2, 0, 3 ]`, tandis que P, vu comme un élément $\mathbf{R}_4[X]$ sera représenté par la liste `[ -5, 2, 0, 3, 0 ]`.

d) Le polynôme nul est particulier car tous ses coefficients sont nuls. En fonction de l'espace vectoriel $\mathbf{R}_n[X]$ dans lequel il est défini, il sera représenté par une liste de $n+1$ coefficients tous nuls : par exemple, dans $\mathbf{R}_3[X]$, il sera représenté par `[0, 0, 0, 0]`.

e) Par convention, on fixe sous python le degré du polynôme nul à -1.

I Degré

■ 64.

Écrire une fonction `degre(P)` qui prend en entrée une liste `p` représentant un polynôme et renvoie le degré de ce polynôme (attention, le degré d'un polynôme n'est pas forcément égal à la longueur de la liste utilisée pour le représenter). Pour le polynôme nul, la fonction doit renvoyer -1 conformément à la convention.

Testez votre fonction avec plusieurs listes, par exemple `[1, 0, 2, 3]` (qui représente $P = 3X^3 + 2X^2 + 1$), ainsi qu'avec des polynômes de degrés inférieurs, et surtout avec le polynôme nul.

II Évaluation

■ 65.

Écrire une fonction `eval(P , x)` qui prend en entrée (une liste représentant) un polynôme `P`, un flottant `x`, et qui renvoie en sortie la valeur de ce polynôme en `x`.

Testez la fonction avec différents flottants et polynômes.

III Somme de deux polynômes

■ 66.

Écrire une fonction `add( P , Q)` qui prend en entrée deux (listes représentant des) polynômes `P` et `Q`, et renvoie une liste représentant leur somme.

La fonction doit pouvoir gérer des polynômes de degrés différents. Par exemple, pour `P = [1, 0, 3]` et `Q = [0, 5, 2, 1]`, le résultat devrait être `[1, 5, 5, 1]`.

Vérifiez votre implémentation en testant la fonction avec plusieurs exemples.

IV Multiplication par un scalaire

■ **67.**

Écrire une fonction `scal_mult(P , alpha)` qui prend en entrée une liste `P` représentant un polynôme P, un flottant `alpha`, et renvoie en sortie une nouvelle liste représentant le polynôme αP. Par exemple, pour `P = [1, 2, 3]` et `alpha = 2`, le résultat doit être `[2, 4, 6]`.

■ **68.**

Rappeler la formule donnant le produit de deux polynômes.

V Produit de deux polynômes

■ **69.**

a) Rappeler la formule donnant le produit de deux polynômes.

b) Écrire une fonction `multiply(P,Q)` qui prend en entrée deux polynômes `P` et `Q`, et renvoie une liste représentant leur produit.

Par exemple, pour `P = [1, 1]` et `Q = [1, -1]`, le résultat devrait être `[1, 0, -1]`.

VI Dérivation

■ **70.**

Écrire une fonction `Deriv(P)` qui prend en entrée un polynôme et renvoie en sortie son polynôme dérivé.

■ **71.**

Écrire une fonction `Deriv(n,P)` qui prend en entrée un entier `n`, un polynôme `P`, et renvoie en sortie son polynôme dérivé d'ordre n, $P^{(n)}$.

VII Égalité de deux polynômes

■ **72.**

Écrire une fonction `is_equal(P,Q)` qui prend en entrée deux polynômes `P` et `Q`, et renvoie vrai si et seulement si les polynômes `P` et `Q` sont égaux Par exemple, pour `P = [1, -1,0,0]` et `Q = [1, -1]`, le résultat devrait être `True`.

Thème 7 : Récursivité

I Présentation de la récursivité

La récursivité est une technique de programmation où une fonction s'appelle elle-même pour résoudre un problème.

Voici des implémentations respectivement itérative et récursive de la fonction factorielle :

```
1 def factorielle_iterative(n):
2     resultat = 1
3     for i in range(1, n + 1):
4         resultat *= i
5     return resultat
```

```
1 def factorielle_recursive(n):
2     if n == 0:
3         return 1
4     else:
5         return n *
              factorielle_recursive(
              n - 1)
```

II Éléments d'une bonne programmation récursive

Pour qu'une fonction soit correctement conçue de manière récursive, il est nécessaire que trois éléments soient parfaitement identifiés et implémentés

a) **Condition d'arrêt** : Il est crucial d'avoir une condition qui termine la récursion (typiquement : une grandeur entière qui atteint sa valeur minimale : 0,1, ou 2).

b) **Appel récursif** : La fonction doit s'appeler elle-même avec une grandeur modifiée, généralement en une valeur strictement inférieure, pour se rapprocher de la condition d'arrêt.

c) **Gestion des cas de base** : Il faut gérer correctement les cas de base pour éviter des appels récursifs infinis.

III Exercices de programmation récursive

■ 73.

Écrire une fonction `arith` qui calcule le n-ième terme d'une suite arithmétique où chaque terme est défini par la relation :

$$a_n = a_{n-1} + d$$

où a_0 est le premier terme et d est la raison.
Arguments d'entrée : un entier n, un entier a_0, un entier d
Sortie : le n-ième terme a_n.

■ 74.

Écrire une fonction nommée `geom` qui calcule le n-ième terme d'une suite géométrique, définie par la relation :

$$u_n = r \cdot u_{n-1}$$

où u_0 est le premier terme et r est la raison de la suite.

Arguments d'entrée :

— Un entier n (l'indice du terme à calculer).

— Un flottant u_0 (le premier terme de la suite).

— Un flottant r (la raison de la suite).

Sortie : Le n-ième terme $g(n)$ de la suite géométrique.

■ 75.

Écrire une fonction `somme_suite` qui calcule la somme des premiers termes d'une suite définie par :

$$S_n = \sum_{k=0}^{n} k$$

Arguments d'entrée : un entier n
Sortie : la somme S_n.

■ 76.

Écrire une fonction `fibo` qui calcule le n-ième terme de la suite de Fibonacci :

$$F_n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ F_{n-1} + F_{n-2} & \text{sinon} \end{cases}$$

Arguments d'entrée : un entier n
Sortie : le n-ième terme F_n.

Utiliser la fonction pour calculer la valeur de F_{38}. Que constate-t-on ?

■ 77.

Écrire une fonction `consec` qui prend en entrée un entier n et renvoie en sortie la liste des entiers de 0 à n.
Arguments d'entrée : un entier.
Sortie : une liste d'entiers.

■ 78.

Écrire une fonction `somme` qui prend en entrée une liste de flottants et renvoie leur somme (fixée à 0 si la liste est vide).
Arguments d'entrée : une liste de flottants
Sortie : la somme des éléments de la liste.

■ 79.

Écrire une fonction `prod` qui prend en entrée une liste de flottants et renvoie le produit de ses éléments. Le produit est fixé à`1` si la liste est vide.
Arguments d'entrée : une liste de flottants
Sortie : le produit des éléments de la liste.

■ 80.

Écrire une fonction `compte` qui prend en entrée une chaîne de caractères et renvoie le nombre de caractères dans cette chaîne.
Arguments d'entrée : une chaîne de caractères
Sortie : le nombre de caractères dans la chaîne.

■ 81.

Écrire une fonction `inverse` qui inverse une chaîne de caractères.
Arguments d'entrée : une chaîne de caractères
Sortie : la chaîne inversée.

■ 82.

Écrire une fonction nommée `est_palindrome` qui vérifie si une chaîne de caractères est un palindrome.
Arguments d'entrée : une chaîne de caractères
Sortie : `True` si la chaîne est un palindrome, sinon `False`.

■ 83.

Écrire une fonction nommée `max_liste` qui trouve le maximum d'une liste non vide d'entiers.
Arguments d'entrée : une liste d'entiers
Sortie : le maximum de la liste.

Thème 8 : Méthodes de tri

Les méthodes de tri jouent un rôle essentiel en informatique pour organiser les données et optimiser leur traitement. Les algorithmes de tri permettent de réarranger les éléments d'une liste selon un ordre donné (croissant ou décroissant). Trois méthodes sont à connaître (parties III, IV et V).

I Modification d'un objet structuré par effet de bord

Sous Python, il est possible qu'une fonction agisse directement sur sa variable d'entrée si c'est un objet mutable [1].

■ **84.**

Tester les deux fonctions suivantes et vérifier si la variable d'entrée a été modifiée ou non par la fonction.

Sans effets de bord

```
def touchepas(L):
    M=L[:]
    M[0]=-15
    return M
```

```
>>L=[1,2,3]
>>L2 = touchepas(L)
```

Avec effets de bord

```
def touche(L):
    L[0]=-15
```

```
>>L=[1,2,3]
>> touche(L)
```

II Méthodes de tri basiques

■ **85.**

Tri naïf

a) Programmer une fonction `pos_min(L)` prenant en entrée une liste `L` de flottants et renvoyant en sortie l'indice dans `L` de son élément minimal.

b) En déduire une fonction `tri_basique(L)` prenant en entrée une liste `L` et renvoyant en sortie la liste `M` obtenue en rangeant les éléments de `L` dans l'ordre croissant.

■ **86.**

Tri d'une liste avec répétitions. *Pour ce tri, on suppose, quitte à d'abord calculer le maximum des éléments de la liste à trier, que les éléments de la liste à trier sont à valeurs dans* $\{1 \dots N\}$*, N étant une constante connue.*

a) Écrire une fonction `multi(L,a)` prenant en entrée une liste `L`, un flottant `a` et renvoyant en sortie le nombre d'occurrences de `a` dans `L`. Par exemple, `multi([1,2,1,3,4,1,2],1)` devrait renvoyer 3, et d'occurrences de `a` dans `L`. Par exemple, `multi([1,2,1,3,4,1,2],6)` devrait renvoyer `0`.

b) Déduire de cette dernière fonction une autre fonction `tri_bis(L)` de tri de liste de flottants.

III Tri à bulles (bubblesort)

Principe. Il repose sur le fait qu'une suite finie $(u_j)_{0\leq j\leq n-1}$ est triée si et seulement si :

$$\forall j \in \{0,\dots,n-2\} \qquad (P_j) : \quad u_j \leq u_{j+1}$$

1. Rappel : parmi les objets structurés mutables on trouve les listes. Sont non mutables en revanche : tuples, chaîne de caractères.

Algorithme.

1. On effectue un parcours des éléments de la liste `L`, et chaque fois que l'on trouve lors de ce parcours deux termes consécutifs `L[j]`, `L[j+1]` ne vérifiant pas la propriété (P_j), on les permute. Cette dernière permutation sera réalisée avec la fonction auxiliaire `permute(L,j)`.
2. Au bout du premier parcours, le plus grand élément de la liste (au moins) est bien trié.

Question : expliquer pourquoi.

3. Ensuite, on en conclut deux choses :

   1) Si la liste est de longueur n, au bout d'un passage, on se retrouve à trier une liste de longueur $n-1$ à l'étape suivante.

   2) Au parcours suivant, il est inutile d'examiner la dernière paire puisque cette dernière n'aura pas à être permutée.

   3) En au plus n parcours, où n est la longueur de la liste, on est certain que la liste est triée.

■ **87.**

a) Écrire une fonction `permute(L,i,j)` prenant en entrée une liste flottants `L`, deux entier `i,j` et qui renvoie en sortie la liste `M` obtenue en échangeant les positions des éléments `L[i]` et `L[j]`.

b) En déduire une fonction `bubbleosrt(L)` prenant en entrée une liste de flottants `L` et modifiant `L` par effet de bord et triée dans l'ordre croissant.

IV Tri insertion

C'est la méthode la plus intuitive de tri, puisque c'est celle que l'on utilise naturellement par exemple pour trier ses cartes dans une partie de belote ou de tarot. Pour autant, elle n'est pas la plus simple à coder.

Algorithme.

a) On suppose qu'à chaque étape de l'algorithme, les éléments situés à gauche d'un élément privilégié de la liste, appelé **pivot** de la liste (et qui sera défini ci-après) sont correctement triés. C'est ce pivot qui sera à insérer correctement pour réaliser le tri.

b) À la première étape, le pivot p est le premier élément de la liste.

c) À chaque étape, on compare le pivot p à l'élément z qui le **précède** dans la liste. Ensuite :

- Si $p < z$, c'est que p est mal placé d'après le point `1.` : donc on le rétrograde dans la liste (à gauche donc) jusqu'à ce qu'il soit en bonne position dans la liste.
- Sinon c'est que p est bien placé.

d) On répète le point `3.` (ce qui constitue une étape), mais avec pour pivot le terme w qui était à droite de p à l'étape précédente.

e) Comme la position du pivot augmente strictement à chaque étape, il se retrouve en dernière position au bout d'un nombre fini d'étapes. D'après `1.`, la liste est triée.

Questions.

a) Écrire une fonction `retrograde(L,i)` qui prend en entrée une liste `L` de flottants, un entier `i`, et qui renvoie en sortie la liste `L` modifiée par effet de bord obtenue en rétrogadant `L[i]` de sorte que le premier élément précédent `L[i]` dans lui soit strictement inférieur.

b) En déduire une fonction `tri_insertion(L)` prenant en entrée une liste de flottants `L` modifiant la liste `L` par effet de bord et triée dans l'ordre croissant.

V Tri rapide (quicksort)

C'est un tri récursif, par définition. Son intérêt est qu'il est plus rapide (en termes de nombre d'opérations) que les autres méthodes de tri qui ont été programmées. Voici une implémentation :

```
1  def qs(L):
2      n  = len(L)
3      if n<=1:
4          return L
5      else:
6          pivot = L[0]
7          Lgauche =[]
8          Ldroite =[]
9          for i in range(1,n
               ):
10             if L[i] <
                   pivot:
11                 Lgauche.
                       append(
                       L[i])
12             else:
13                 Ldroite.
                       append(
                       L[i])
14     return qs(Lgauche)+[
           pivot]+qs(Ldroite)
```

■ 88.

Expliquer son principe.

VI Applications des méthodes de tri

■ 89.

Médiane d'une statistique univariée

Définition. Soit $x = (x_1 < \cdots < x_n)$ une série statistique de longueur n (triée). La médiane Q_2 de la série est :

$$Q_2 = \begin{cases} x_{1+\frac{n-1}{2}} & \text{si n est impair} \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{1+\frac{n}{2}}\right) & \text{si n est pair} \end{cases}$$

Programmer une fonction `mediane(L)` prenant en entrée une liste de flottants `L` représentant une série statistique et donnant en sortie la valeur de sa médiane (attention à la numérotation des éléments d'une liste en python)

■ 90.

Tri d'une liste en fonction d'une autre

a) Écrire une fonction `tri_couples(L)` prenant en entrée une liste `L` de 2-tuples (a, b) tels que a est un flottant, et renvoyant en sortie la liste `M` obtenue en rangeant les éléments de `L` par ordre croissant des a.

b) En déduire une fonction `tri2(L1,L2)` prenant en entrée deux listes de même longueur `L1`, `L2`, la liste `L1` étant une liste de flottants, et renvoyant en sortie la liste `L2` triée par ordre croissant des éléments de `L1`.

Thème 9 : Variables aléatoires à densité

Il est souhaitable de disposer du formulaire python du concours Agro-Veto, de son cours sur le sujet, et de papier pour calculer !

I Lois usuelles

■ 91.

a) Écrire une fonction `Unif(a,b)` dont l'exécution renvoie en sortie une simulation d'une variable aléatoire à densité d'une loi uniforme sur le segment $[a,b]$, c'est-à-dire que l'appel de `Unif(a,b)` renvoie en sortie un flottant tiré au hasard entre a et b. *Indication : utiliser que si $k>0$, `k*random()` renvoie un flottant au hasard entre 0 et k, et que `a+random()` renvoie au hasard un flottant entre a et a+1.*

b) Écrire une fonction `expo(mu)` dont l'exécution renvoie en sortie une simulation d'une variable aléatoire à densité de loi exponentielle de paramètre μ. *Rappeler ce qu'est la densité exponentielle, et montrer que si U suit une loi uniforme sur $[0,1]$, la variable $V=-\frac{1}{\mu}\ln U$ suit la loi exponentielle de paramètre μ.*

c) Écrire une fonction `normale(mu,sigma)` dont l'exécution renvoie en sortie une simulation d'une variable aléatoire à densité de loi normale de paramètres μ,σ^2. *Utiliser le formulaire*

II Exercices

■ 92.

a) Si X suit une loi exponentielle de paramètre $\lambda>0$ et Y suit une loi exponentielle de paramètre $\mu>0$, simuler la variable $Z=\min(X,Y)$.

b) Calculer la loi de Z mathématiquement.

c) Montrer que la loi de Z est une loi exponentielle de paramètre α à préciser.

d) Constater en traçant les histogrammes des fréquences pour la variable Z et une variable de loi $\mathscr{E}(\alpha)$ la correction de votre réponse à `3.`

■ 93.

Soit $\lambda>0$ un réel, et f la fonction définie par : $f(x)=\begin{cases}0 & \text{si} \quad x\leq 1\\ \dfrac{\lambda}{x\sqrt{x}} & \text{si} \quad x>1.\end{cases}$

a) Déterminer λ tel que f soit une densité de probabilité d'une variable aléatoire X.

b) Déterminer la loi de la variable alétaoire $Y=\ln X$.

c) En déduire une méthode de simulation de la variable Y.

■ 94.

Soit $a>0$ un réel et f_a la fonction définie sur $\mathbf{R}$ par $f_a(x)=\dfrac{a}{\pi(x^2+a^2)}$.

a) Montrer que f_a est une densité.

b) Montrer que si U suit une loi uniforme sur $]0,1[$, alors la variable $Z=\tan\left(\pi U+\dfrac{\pi}{2}\right)$ est une variable à densité de densité f_1.

c) Simuler une variable aléatoire X de loi f_1.

d) Estimer informatiquement plusieurs fois l'espérance de la variable X. La variable X admet-elle une espérance ?

e) Soit $a>0$. Simuler une variable de loi f_a.

■ 95.

Soit $a\in\mathbf{R}$, $b>0$ et $f_{a,b}$ la fonction définie sur $\mathbf{R}$ par :

$$f_{a,b}(x)=\frac{1}{2b}\exp\left(-\frac{|x-a|}{b}\right).$$

a) Montrer que $f_{a,b}$ est une densité.

b) Soit U une variable aléatoire de loi exponentielle de paramètre 1. Soit V une variable aléatoire de loi uniforme sur l'ensemble fini $\{-1,1\}$. On suppose que U et V sont indépendantes. Montrer que la variable $X = a + bUV$ est à densité et admet pour loi la densité $f_{a,b}$.

c) Simuler la variable X de la question précédente.

d) Estimer empiriquement l'espérance et la variance de X.

Thème 10 : Exercices supplémentaires

■ 96.

Soit $n \geq 1$ un entier et L une liste d'entiers à valeurs dans $\{0, \ldots, n-1\}$.

Écrire une fonction `compte_occurences(L)` qui prend la liste `L` en entrée et renvoie en sortie la liste `M` telle que `M[i]` soit égale au nombre d'occurrences de l'entier `i` dans `L`. Par exemple, pour `L =[1,0,3,3,2,1,3,4]`, `compte_occurences(L)` renvoie `[1,2,1,3,1]`

■ 97.

a) Programmer une fonction `noccur(m,x)` prenant en entrée une chaîne de caractères `m`, un caractère `x` et renvoyant en sortie le nombre d'occurences de la lettre `x` dans le mot `m`.

b) On considère un mot codé `m` codé de la manière suivante : chaque lettre du mot `m` est décalée de p unités dans l'alphabet (en rebouclant à partir de a après z), où p est le nombre d'occurrences de `x` dans le mot initial. Ainsi dans le mot «bananez», le a est remplacé par c, et z est remplacé par a. Programmer une fonction `decode(m)` qui décode un mot codé par le procédé énoncé. On pourra utiliser la commande `c.find(k)` qui donne l'indice de la première occurence du caractère `k` dans la chaîne `c`.

■ 98.

Soit X une variable aléatoire finie de loi donnée par sa loi :

x_0	x_1	$\ldots$	x_{n-1}
p_0	p_1	$\cdots$	p_{n-1}

où $p_k = P(X = x_k)$ et $X(\Omega) = \{x_0 \ldots x_{n-1}\}$.

a) Programmer une fonction `esperance(L)` où `L` est une liste de la forme : `L=[[x0,p0],[x1,p1],...,[xn-1,xn]]`, et renvoyant en sortie l'espérance de la variable X.

b) Programmer une fonction `verif(L)` prenant en entrée une liste `L`, et renvoyant en sortie `True` si la liste `L` est bien la loi d'une variable aléatoire finie, et `False` sinon (les éléments de `L` doivent être des listes de deux flottants, les `xi` doivent être distincts deux à deux, les `pi` sont positifs et de somme égale à `1`).

■ 99.

a) Programmer une fonction qui prend en entrée une chaîne de caractères `m`, et qui renvoie `True` si `m` contient autant de «(» que de «)», et `False` sinon.

b) Programmer une fonction qui vérifie pour la chaine `m` la condition précédente mais également la condition suivante : à tout rang dans la chaîne `m`, le nombre de parenthèses ouvrantes est supérieur ou égal au nombre de parenthèses fermantes.

■ 100.

Soit L une liste de `0` et de `1`. Programmer une fonction `serie1(L)` qui renvoie la longueur des deux premières séries de `L`. Par exemple `serie1([0,0,0,1,1,0])` renvoie `3,2` car la première série est de longueur 3 et la deuxième de longueur 2.

■ 101.

Chaque bactérie d'une population donnée de N bactéries à une proabilité p de mourir au bout d'une heure. Les comportements des bactéries sont supposés mutuellement indépendants.

a) Programmer une fonction `evolution(N,p)` renvoyant le nombre de bactéries vivantes au bout d'une heure.

b) Programmer une fonction `lifetime(N,p)` renvoyant le nombre d'heure minimal à partir duquel toute la population bactérienne est éradiquée.

c) On ajoute une propriété au modèle : chaque bactérie a une probabilité p de mourir et $q = 1-p$ de se dupliquer. Ces deux derniers évènements sont supposés incompatibles. Progammer une fonction `evolution2(N,p,n)` qui renvoie la liste des effectifs de la population bactérienne au bout de `n` heures.

■ 102.

Un polynôme P est représenté par la liste de ses coefficients rangés par degré croissant.

a) Programmer une fonction `eval(L,a)` qui prend en entrée une liste `L` représentant un polynôme P, un flottant `z` et renvoyant la valeur de $P(z)$.

b) La méthode de Hörner consiste à évaluer $P(z)$ en minimisant le mombre d'opérations de la façon suivante : pour $P = a_0 + a_1X + \cdots + a_nX^n$, on écrit $P - a_0 = XP_1$, où $P_1 = a_1 + a_2X + \cdots + a_nX^{n-1}$, et on recommence sur P_1 : $P_1 - a_1 = XP_2$, et ainsi de suite. De façon générale, si on pose : $P_0 = P$, on a donc de proche en proche : $P_k = a_k + XP_{k+1}$, et $P_{n-1} - a_{n-1} = Xa_n$. Programmer le calcul de $P(z)$ par la méthode de Hörner. Intérêt de la méthode ?

■ 103.

a) Écrire une fonction prenant en entrée une liste de flottants et renvoyant la liste des indices du maximum de la liste.

b) Écrire une fonction qui prend entrée une liste `L` et un flottant `a` et qui renvoie en sortie le rang de la première occurrence de `a` dans `L` et sa dernière occurrence. On traitera les cas particuliers où `a` est absent de la liste ou le cas où il n'apparaît qu'une seule fois.

■ 104.

a) Programmer une fonction `difference`, qui prend en entrée deux listes `L1` et `L2` de même longueur et renvoie le nombre de fois que `L1[i]` et `L2[i]` sont différents.

b) Programmer une fonction qui prend en entrée deux chaines de caractères, `txt` et `seq`, renvoyant `True` si `seq` est une sous-chaîne de `txt`, et `False` sinon.

■ 105.

Soit `L` une liste et T un entier strictement positif. On dit que `L` est T-périodique si `L[i+T]=L[i]`.

a) programmer une fonction `periode(L,T)` qui renvoie `True` si la liste `L` est T-périodique et `False` sinon.

b) Soit `L` une liste périodique. Programmer une fonction qui calcule la plus petite période de `L`.

■ 106.

a) Programmer une fonction `tx_w(L)` qui prend en entrée une liste de flottants `L`, et calcule la proportion de `-1` dans `L`.

b) Soit $N \geq 2$. Un scrutin doit élire un candidat parmi N candidats $c_1, \ldots, c_N$. Les suffrages exprimés sont consignés dans une liste `L` définie de la façon suivante : `L[0]` contient le nombre de votes blanc ou nul, et pour `j`≥ 1, `L[j]` contient le nombre de votes en faveur de c_j. Programmer une fonction `est_elu(L,j)` qui prend en entrée une liste de suffrages `L`, un entier `j` et qui renvoie `True` si c_j obtient la majorité absolue (strictement plus de la moitié des votes ni blancs, ni nuls), et `False` sinon.

■ 107.

a) Programmer une fonction `diff01(L)` prenant en entrée une liste de flottants `L`, et renvoyant la liste des éléments de `L` dans le même ordre en omettant les `0` et les `1`.

b) Programmer une fonction `diff(L,k)` prenant en entrée une liste de flottants `L`, un entier `k`> 0, et renvoyant la liste `M` égale à liste `L` dans laquelle tous les éléments en position multiple de `k` sont remplacés par des `0`.

■ 108.

a) Programmer une fonction `compte(L,x)` qui calcule le nombre d'occurrences du flottant `x` dans la liste `L`.

b) Programmer une fonction `binomial(m,N,p)` prenant en entrée des entiers `m`,`N`≥ 1, un flottant `p`$\in]-0;1[$, et renvoyant en sortie une liste de `m` simulations de la loi $\mathscr{B}(N,p)$.

c) Programmer une fonction `simuBinomial(m,N,p)` prenant en entrée des entiers `m`,`N`≥ 1, un flottant `p`$\in]-0;1[$, et renvoyant en sortie une liste `M` de longueur `N+1` telle que `M[j]` est égal au nombre de fois que l'on a observé exactement `j` succès lors de `m` simulations de la loi $\mathscr{B}(N,p)$.

■ 109.

a) Écrire une fonction d'argument une liste L qui teste si la liste L vérifie l'hypothèse $\mathscr{H}$ suivante :
$\mathscr{H}$: les éléments de la liste L sont des entiers qui sont compris au sens large entre 0 et $n-1$ avec n la longueur de la liste L.

b) Écrire une fonction d'argument une liste L vérifiant l'hypothèse $\mathscr{H}$ qui teste si la liste L vérifie l'hypothèse $\mathscr{H}'$ suivante :
$\mathscr{H}'$: la liste L contient exactement une fois chaque valeur entre 0 et $n-1$ où n la longueur de la liste L.

■ 110.

Soit a et b deux entiers naturels avec $a < b$.

a) Écrire une fonction `verif(`L,a,b`)` qui a pour paramètres une liste L d'entiers naturels et 2 entiers a et b et qui renvoie `True` si tous les éléments de L sont dans l'intervalle $[\![a,b]\!]$ et `False` sinon.

b) Soit L une liste dont chaque élément est dans l'intervalle d'entiers $[\![a,b]\!]$. Écrire une fonction `denombre(`L,a,b`)` qui détermine le nombre d'apparitions de chacune des valeurs possibles (entre a et b) de la liste L et qui renvoie le résultat dans une liste dont le premier élément représentera le nombre de a dans L, le deuxième le nombre de $a+1$ dans L etc.

Exemple : pour $a = 0$ et $b = 4$, `denombre(`$[1,3,0,4,1,3,1],0,4$`)` renverra $[1,3,0,2,1]$.

■ 111.

On souhaite exploiter le suivi d'une randonnée en effectuant des mesures lors de différents points de passage : latitude, longitude, altitude et temps (date).
Ces données sont contenues dans une liste `coords`, où `coords[i]` désigne la liste des 4 mesures effectuées au point de passage numéro `i`. Ainsi `coords[i][2]` renvoie par exemple l'altitude relevée au point de passage numéro `i`.

a) Écrire une fonction `temps(coords)` qui renvoie la liste des temps relevés lors de la randonnée `coords`.

b) Sans utiliser la fonction `max`, écrire une fonction `plus_haut(coords)` qui renvoie la liste `[lat, long]` du point le plus haut lors de la randonnée `coords`.

■ 112.

a) Écrire une fonction Python qui simule une série de N lancers d'une pièce équilibrée, et qui renvoie la liste des résultats de ces lancers ("Pile" est codé par 1, et "Face" par 0).

b) Écrire une fonction Python qui simule une série de lancers d'une pièce équilibrée jusqu'à l'obtention de la configuration "Pile, Pile, Face", et qui renvoie le nombre de lancers nécessaires à l'apparition de cette configuration. À l'aide de cette fonction, évaluer le temps moyen d'attente de cette configuration.

■ 113.

Un entier naturel non nul n est dit « parfait » s'il est égal à la somme de ses diviseurs autres que lui-même. On rappelle que pour k tel que $1 \leqslant k \leqslant n$, k divise n, si et seulement si `n % k` est nul.

a) Écrire une fonction `s(n)` qui renvoie la somme des diviseurs de n autres que lui-même.

b) Écrire une fonction `prop_parfait(N)` qui renvoie la proportion de nombres parfaits compris entre 1 et N.

c) Question supplémentaire qui n'était pas sur la feuille : comment faire pour renvoyer le k-ieme nombre parfait ?

■ 114.

a) Écrire une fonction `sans_rep(L)` qui prend en argument une liste L et qui renvoie la liste M des éléments distincts de L.

b) Écrire une fonction `list_oc(L)` qui renvoie les listes M et O, où O est la liste contenant le nombre d'occurrences des éléments de la liste M dans cet ordre.

■ 115.

a) Écrire une fonction qui prend en entrée une chaîne de caractères s et une liste de caractères Lc et qui renvoie l'indice à partir duquel l'enchainement de caractères de Lc donne s s'il existe, et sinon -1.
Exemple : `Lc = ['a','b','c','d']` Si `s='cd'`, la fonction renvoie 2. Si `s= 'ad'`, la fonction renvoie -1.

b) Écrire une fonction qui prend en argument une matrice M, deux entiers i et j et un vecteur (x, y) et qui renvoie la liste de tous les éléments de M rencontrés à partir de la position (i, j) en suivant un déplacement (x, y).

■ 116.

a) On considère une colonies de N bactéries. Chaque heure, les bactéries peuvent mourir avec une probabilité p.

1) Écrire une fonction `evolution` qui prend en argument l'entier `N` et le flottant `p` et qui renvoie le nombre de bactéries vivantes après une heure.

2) Écrire une fonction `extinction` qui prend en argument l'entier `N` et le flottant `p` et qui détermine au bout de combien d'heures toutes les bactéries sont mortes.

b) Mêmes questions si on suppose que les bactéries peuvent également se diviser avec une probabilité de q telle que $0 < p + q < 1$.

■ 117.

a) Écrire une fonction `f(L,n,p)` qui prend en argument une liste de couples d'entiers `L` deux entiers naturels `n` et `p`, et qui renvoie la liste des couples `(a,b)` de `L` vérifiant les conditions : $0 \leqslant a < n$ et $0 \leqslant b < p$.

b) Écrire une fonction `voisins(M,i,j)` qui prend en argument une matrice `M` de taille minimale $(3,3)$, des entiers `i` et `j` correspondant à un coefficient de la matrice M et qui renvoie la liste constituée de `M[i,j]` ainsi que les coefficients situés dans les au plus 8 cases adjacentes.

■ 118.

a) Écrire une fonction `somme(L,i,j)` qui prend en argument une liste L et deux entiers i et j tel que $i \leqslant j < len(L)$ et qui renvoie la somme des termes d'indice i inclus à j exclu.

b) Écrire une fonction `L_sous(L,k)` qui prend en argument une liste L et un entier k tel que $k < len(L)$ et qui renvoie la sous-liste de L de longueur k dont la somme des termes est maximale.

■ 119.

Partant d'un point A du plan de coordonnées entières, on se déplace dans le $\mathbf{Z}^2$ selon les quatre directions : à droite (`'d'`), à gauche (`'g'`), en bas (`'b'`) et en haut (`'h'`).

a) Écrire une fonction `prochain(A,dir)` qui prend en argument un couple `A` et une direction `dir` et renvoie le point d'arrivée en partant de `A` et en suivant la direction `dir`.

b) Écrire une fonction `chemin(mot)` qui renvoie la liste des couples du chemin partant de $(0,0)$ en empruntant les directions données par `mot`.

c) Écrire une fonction `contour(mot)` qui renvoie `True` si le chemin a fait une boucle sans se croiser et `False` sinon.

■ 120.

a) Écrire une fonction `LongCr1` prenant en entrée une liste L et qui renvoie la longueur de la première sous-liste croissante au sens large de `L`.
Exemple : `L=[0,1,2,1,2,3,0]`, la fonction renvoie 3.

b) Écrire une fonction `longCr` prenant en entrée une liste L et qui renvoie une liste contenant la longueur de toutes les sous-listes croissantes au sens large de L.
Exemple : `L=[0,1,2,1,2,3,0]`, la fonction renvoie `l=[3,3,1]`.

c) Question supplémentaire : expliquer à l'oral comment modifier la dernière fonction `longCr` afin qu'elle renvoie aussi les sous-listes de `L` sous forme de liste.

■ 121.

a) Écrire une fonction `voyelles` qui prend en entrée une chaîne de caractère S et renvoie `True` si elle contient au moins une voyelle, `False` sinon.

b) Soit `P` une fonction qui prend en argument un caractère et renvoie un booléen. Écrire une fonction `test(S,P)` avec S une chaîne de caractères et qui renvoie `True` si au moins un caractère de S évalué par P renvoie `True`, `False` sinon.

c) Question supplémentaire : variante de la question 2 avec LS liste de chaînes de caractères, et LP liste de fonctions qui renvoie `True` si au moins une fonction `test(S,P)` renvoie `True`.

■ 122.

a) Écrire une fonction `extrema` qui prend en argument une liste de nombres entiers `L` et qui renvoie le minimum et le maximum de la liste sans utiliser les fonctions min et max.

b) Écrire une fonction `occurrences` qui prend en argument une liste de nombres entiers `L` et qui renvoie sous forme de liste le nombre d'occurrences dans `L` de tous les nombres compris entre le min et le max.
Exemple : pour la liste `[0,2,5,8,-1,2,6]`, la fonction renvoie `[1,1,0,2,0,0,1,1,0,1]`.

■ 123.

Si $A=(a_0,a_1,\dots,a_{n-1})$ est une suite d'entiers naturels et si P est une partie de $\{0,1,\dots,n-1\}$, on note $s(A,P)$ la somme des a_i pour i appartenant à P. Par exemple, si $n=8$ et $P=\{2,3,6\}$, $s(A,P)=a_2+a_3+a_6$.
Une partie P de $\{0,1,\dots,n-1\}$ sera représentée par une liste de booléens de longueur n, $P[i]$ prenant la valeur `True` si et seulement si i appartient à P. Ainsi, dans l'exemple précédent, P est représentée par la liste `[False,False,True,True,False,False,True,False]`.

a) Écrire une fonction `somme` de paramètre `A` et `P` qui renvoie la valeur de $s(A,P)$.

b) On souhaite parcourir toutes les parties de $\{0,1,\dots,n-1\}$.

1) Écrire une fonction `suivant(P)` où P est une liste représentant une partie de $\{0,1,\dots,n-1\}$ et qui modifie P de la façon suivante :

— on parcourt `P` en partant de la gauche et tant que $P[i]$ prend la valeur `True`, on modifie cette valeur en `False`

— si on arrive à un indice i tel que $P[i]$ prend la valeur `False` , on modifie cette valeur en `True`, on arrête le calcul et on renvoie le booléen `True`,

— si P ne contenait que des `True`, on renvoie le booléen `False`.

Par exemple, si on applique `suivant` à `P=[True,True,False,True,False,False,True,False]`, P sera modifiée en :
`[False,False,True,True,False,False,True,False]`,
et la fonction renvoie le booléen `True`.

2) Si on initialise P à la valeur `[False,...,False]`, on pourra parcourir toutes les parties de $\{0,1,\dots,n-1\}$ en appliquant suffisamment de fois la fonction `suivant`.
Écrire une fonction `listeP(n)` qui renvoie la liste de parties de $\{0,\dots,n-1\}$ en utilisant la fonction suivant.

■ 124.

On rappelle qu'un brin d'ADN est composé de 4 nucléotides A,T,C,G. On définit $N=['A','T','C','G']$. L'ADN a une structure en double hélice où l'on retrouve une complémentarité des 2 brins, telle que A est complémentaire de T et C est complémentaire de G.

a) Définir une fonction `brin` qui crée un brin représenté par une liste de longueur n dans lequel chaque nucléotide est choisi uniformément dans N.

b) Écrire une fonction `brin_complementaire` prend en argument un brin et qui renvoie le brin complémentaire.

c) Écrire une fonction qui pour un brin donné renvoie la fréquence de chaque nucléotides.

d) Écrire une fonction qui prend en argument n et p et qui permet d'obtenir un brin de taille n où A et T apparaissent avec probabilité p, G et C apparaissent avec probabilité $1/2-p$, $0<p<1/2$.

■ 125.

a) Écrire une fonction qui prend en argument un réel $p\in]0,1[$ et qui simule une variable qui suit la loi géométrique de paramètre p.

b) On lance N dés à 6 faces, dès qu'un dé donne 6, on le retire et on poursuit jusqu'à ce que tous les dés aient donné 6. Écrire une fonction qui simule le jeu et renvoie le nombre de tours.

■ 126.

On considère une grille de jeu de taille $n\times n$. Chaque case à pour valeur une simulation de loi géométrique de paramètre $p\in]0,1[$.

a) Ecrire une fonction permettant de créer une grille de jeu.

b) Lors de la partie, on démarre de la case en haut à gauche et on souhaite arriver à la case en bas à droite. Le pion ne peut se déplacer que vers le bas ou la droite (tant qu'il ne sort pas de la grille). Il se déplace vers le bas uniquement si la case à sa droite indique une valeur strictement plus grande ou si il n'y a plus de case à droite. Écrire une fonction qui renvoie une liste qui recense chacun des déplacements noté 'b' vers le bas ou 'd' à droite.

■ 127.

a) Écrire une fonction `permut(n)` qui prend en argument un entier n et qui renvoie une permutation des entiers de 0 à $n-1$ sous forme d'une liste.

b) Écrire une fonction `EstUnDerangement(Perm)` qui renvoie `True` si la liste `Perm` est un dérangement (aucun entier n'est à la bonne place) et renvoie `False` sinon.

■ 128.

a) Écrire une fonction prenant en argument une liste et qui renvoie la liste triée en utilisant une méthode de votre choix.

b) Écrire une fonction `valeurs_et_occ` qui prend en argument une liste L déjà triée et qui renvoie deux listes : V qui contient les différentes valeurs présentent dans la liste L et E les occurrences de ces valeurs.
Exemple : pour
`L=[1,1,2,2,2,3,3,3,3,3,4,4,5,5,5,5,5,5,5,6]`,
on obtient `V=[1,2,3,4,5,6]` et `E=[2,3,5,2,7,1]`.

■ 129.

a) Écrire une fonction `el_commun(L1,L2)` d'arguments deux listes qui renvoie `True` si ces deux listes ont au moins un élément commun et `False` sinon.

b) Écrire une fonction `Liste_commun(L1,L2)` d'arguments deux listes qui renvoie la liste des éléments communs à ces deux listes.

■ 130.

a) Écrire une fonction `test_arithmetique(L)` d'argument une liste de flottants de longeur au moins 3 qui renvoie `True` si la liste L est constituée des termes d'une suite arithmétique et `False` sinon.

b) Même question avec une suite géométrique.

■ 131.

a) Écrire une fonction `Maxi(Tab)` d'argument une liste d'entiers positifs qui renvoie le maximum de la liste Tab sans utiliser la fonction `max` de Python.

b) Écrire une fonction `Occur(Tab)` d'argument une liste `Tab` d'entiers positifs qui renvoie la liste des occurences dans `Tab` de tous les entiers compris entre 0 et le maximum de `Tab`.
Par exemple, si `Tab=[9,2,4,3,2,3,7,5,3,6]` alors la fonction doit envoyer la liste `[0,0,2,3,1,1,1,1,0,1]`.

■ 132.

a) Écrire une fonction `Test(ADN)` d'argument une chaîne de caractères *ADN*, qui renvoie `True` si *ADN* est une molécule d'ADN (c'est-à-dire qui ne contient que les lettres *A*, *T*, *C* ou *G*) et `False` sinon.

b) Écrire une fonction `Mut(ADN)` d'argument une chaîne de caractères *ADN* qui renvoie la chaîne dans laquelle le caractère *T* a été remplacé par le caractère *U*.

c) Un gène est une chaîne de caractères *A*, *U*, *C*, *G* délimité par un codon start et un cadon stop. Les codons start sont *AUG*, *GUG* et *UUG*. Les codons stop sont *UAA*, *UAG* et *UGA*.

Écrire une fonction `Test_Gene(ARN)` d'argument une chaîne de caractères *ARN* qui renvoie `True` si *Gene* est un gène et `False` sinon.

■ 133.

a) Écrire une fonction `Maxi2(L)` d'argument une liste L d'entiers qui renvoie le deuxième plus grand entier de L, sans utiliser la fonction `max` de Python.

b) Écrire une fonction `Indice(L,x)` d'argument une liste L d'entiers et un entier x qui renvoie -1 si x n'est pas dans L et le rang de x dans L si L était rangée dans l'ordre croissant sinon. On ne cherchera pas à trier la liste L.

■ 134.

a) Écrire une fonction `Liste_multi(n)` d'argument un entier n qui renvoie la liste des entiers de 1 à n autant de fois que leur valeur. Par exemple `Liste_multi(3)` doit renvoyer la liste $[1,2,2,3,3,3]$.

b) On tire une boule au hasard dans une première urne contenant 5 boules numérotées de 1 à 5.

Si on a tiré la boule numérotée n, on place dans une deuxième urne des boules numérotées comme défini par la fonction `Liste_multi(n)`.

Écrire une fonction qui simule l'expérience et renvoie le numéro de la boule tirée dans la deuxième urne.

c) Estimer sous Python la probabilité de tirer dans la deuxième urne la boule numérotée 1.

Calculer mathématiquement cette probabilité.

■ 135.

a) Écrire une fonction `Entre(L)` prenant en entrée une liste L d'entiers qui renvoie `True` si tous les éléments de L sont compris entre 0 et n où n le nombre d'éléments de L et `False` sinon.

b) Écrire une fonction `Recherche(x,L)` prenant en argument un nombre x et une liste L et renvoyant la liste des indices de x dans L si x est dans L et la liste vide si x n'est pas dans L.

Par exemple si $L = [1,5,6,1]$ alors `Recherche(1,L)` renvoie $[0,3]$).

■ 136.

On appelle nuage de points une liste de listes de longueur 2 correspondant aux coordonnées d'un point dans le plan. Exemple de nuage de n points : $[[x_1,y_1],[x_2,y_2],\cdots,[x_n,y_n]]$.

a) Écrire une fonction `distance(A,B)` qui prend en arguments 2 listes A et B de coordonnées de 2 points et renvoie la distance entre ces 2 points.

b) Écrire une fonction `point_moyen(Nuage)` qui prend en argument une liste nuage de points et renvoie la liste des coordonnées du point moyen : $[\overline{x},\overline{y}]$.

c) Écrire une fonction `plus_proche_G(Nuage)` qui renvoie le point le plus proche du point moyen.

d) Écrire une fonction `representation(Nuage)` qui représente ce nuage de points, son point moyen et le point le plus proche du point moyen.

■ 137.

a) Écrire une fonction `Freq(ADN)` qui prend en argument une chaîne de caractères `ADN` constituée uniquement des lettres *A*, *T*, *C*, *G* et renvoie la liste des fréquences de chaque nucléotide.

b) Écrire une fonction `Long1(ADN)` qui prend en argument une chaîne de caractères `ADN` et qui renvoie la liste du nombre de répétitions successives de la première nucléotide et la nucléotide.

Par exemple, si `ADN = 'AAAGGTTCACGTG'` alors `Long1(ADN)` renvoie `[3,'A']`.

c) Écrire une fonction `Brin(ADN)` qui prend en argument une chaîne de caractères `ADN` et qui renvoie la liste constituée du nombre de répétitions successives de chaque nucléotide suivi de la nucléotide.
Par exemple, si `ADN='AAAAGGTTTCCC'` alors `Brin(ADN)` renvoie :
`[4,'A',2,'G',3,'T',3,'C']`.

■ 138.

On étudie des séries binaires, composées de 0 et de 1, comme par exemple : 0100111011000110 que l'on représente par des listes composées de 0 et de 1.
On cherche à compresser une série binaire. Pour cela, les suites de chiffres similaires successifs sont représentés dans une liste par le nombre de chiffres dans la suite binaire. Par exemple la série 001011100110 donne le vecteur $[0\,;\,2\,;\,1\,;\,1\,;\,3\,;\,2\,;\,2\,;\,1]$, le premier 0 signifiant que la série commence par un 0 (et donc pouvant être remplacé par un 1 ayant la même signification).

a) Écrire une fonction permettant cette compression.

b) Sur le même modèle que précédemment, écrire une fonction permettant de décompresser une série commençant par un 0 ou un 1, et composée ensuite de chiffres entre 1 et 9.

c) Écrire une fonction qui dans une telle série binaire remplace les 1 de la plus longue suite de 1 successifs par des chiffres 2.

■ 139.

a) Écrire une fonction `Eval(P,a)` qui prend en argument la liste des coefficients du polynôme P dans la base canonique et qui renvoie $P(a)$

b) Écrire une fonction `Deriv(P)` qui prend en argument la liste P et qui renvoie la liste des coefficients de son polynôme dérivé P'.

c) Écrire une fonction `f(P)` qui prend en argument la liste P et qui renvoie la liste des coefficients de $f(P) = 2XP - P'$

■ 140.

a) Écrire une fonction `listeX(n)` d'argument un entier strictement positif n qui renvoie une subdivision de l'intervalle $[0,1]$ en n segments de même largeur, c'est-à-dire la liste $[x_0, x_1, ..., x_n]$ des bornes de n segments de même largeur de l'intervalle $[0,1]$ (en particulier : $x_0 = 0$ et $x_n = 1$).

b) On considère l'équation différentielle avec condition initiale (E) : $y' = xy$ et $y(0) = 1$.
On rappelle que la méthode d'Euler pour résoudre numériquement l'équation (E) consiste à remplacer sur chaque intervalle $[x_k, x_{k+1}]$, la courbe représentative d'une solution y par sa tangente en x_k. On définit ainsi une suite $(y_k)_k$ où $y_0 = y(0)$ et y_{k+1} est l'ordonnée du point d'abscisse x_{k+1} de cette tangente.
Écrire une fonction `Euler(n)` d'argument un entier strictement positif n qui renvoie les listes $[x_0, x_1, \cdots, x_n]$ et $[y_0, y_1, \cdots, y_n]$.

c) Représenter sous Python et sur un même schéma la solution numérique et la solution exacte de (E).

■ 141.

a) Soit a et b deux réels et f une fonction continue sur $[a,b]$. Pour tout entier strictement positif n, on pose $R_n = \dfrac{b-a}{n}\displaystyle\sum_{k=0}^{n-1} f\left(a + k\frac{b-a}{n}\right)$.
Écrire une fonction `R(f,a,b,n)` d'arguments une fonction f, deux flottants a, b et un entier n qui renvoie R_n.

b) Calculer sous Python une valeur approchée de l'intégrale $I = \dfrac{1}{2\pi}\displaystyle\int_{-2}^{2} \sqrt{4-x^2}\,\mathrm{d}x$.

■ 142.

a) Écrire une fonction `el_com(G,T)` prenant pour argument deux listes G et T et qui renvoie le nombre d'éléments communs aux deux listes.

b) Écrire une fonction `loto()` qui renvoie un tirage du loto, c'est-à-dire 6 tirages successifs sans remise de 6 boules dans une urne contenant des boules numérotées de 1 à 49.

c) Hugo est né le 26 avril 1998 et joue toujours les mêmes 6 numéros 260498.
Écire une fonction `Nb_Gagnant(T)` d'argument un tirage du loto (6 numéros) et qui renvoie le nombre de numéros gagnants pour Hugo.

d) Écrire une fonction `Gagne(n)` d'argument un entier strictement positif n qui renvoie le nombre de fois où Hugo gagne (6 numéros gagnants) en jouant n fois au loto.

Comment évaluer numériquement la probabilité pour Hugo de gagner au loto ?

■ 143.

On considère des listes composées uniquement de 0 et de 1. Une liste est dite mixte lorsqu'elle contient au moins un 0 et au moins un 1.

a) Écrire une fonction `est_mixte(L)` d'argument une liste L de 0 et de 1 renvoyant `True` si la liste est mixte est `False` sinon.

b) Écrire une fonction `plus_long_suite1(L)` d'argument une liste L de 0 et de 1 renvoyant le plus grand nombre de 1 consécutifs dans L s'il y en a au moins 1 et 0 sinon.
Par exemple si $L = [1,0,1,1,1,0,0,1,1]$, `plus_long_suite1(L)` doit renvoyer 3.

■ 144.

a) Écrire une fonction `somme(L,i,j)` prenant pour argument une liste L de flottants et des entiers naturels i et j qui renvoie la somme des valeurs des éléments de L dont les indices sont compris entre i inclus et j exclus.

b) Écrire une fonction `sous_liste_long(L,k)` prenant pour argument une liste L et un entier k (inférieur à la longueur de L) qui renvoie la sous-liste de L de longueur k dont la somme des termes est maximale.

■ 145.

a) Écrire une fonction qui permet de simuler une variable aléatoire qui suit la loi géométrique de paramètre p ($p \in]0,1[$).

b) En déduire une fonction qui permet de simuler la variable aléatoire $Y_n = max(X_1,\dots,X_n)$ où $X_1,\dots,X_n$ sont des variables aléatoires indépendantes qui suivent la loi géométrique de paramètre p.

■ 146.

On s'intéresse dans cet exercice à des listes d'entiers. L'utilisation de `max` ou de `count` est interdite.

a) Écrire une fonction prenant en argument une liste d'entiers L et un entier $k \in \mathbb{N}$. Cette fonction renvoie `True` si tous les entiers de cette liste sont compris entre 0 et k et `False` sinon.

Par exemple, pour `L=[0,2,0]`, la fonction renvoie `True` si $k = 2$ ou $k = 3$ et `False` si $k = 1$.

b) Écrire une fonction de mêmes arguments, où la liste est supposée ne contenir que des entiers compris entre 0 et k. Cette fonction renvoie l'élément le plus fréquent (ou l'un d'entre eux s'il y en a plusieurs).
Par exemple, pour `L=[0,4,0,1,4]` et $k = 3$, la fonction renvoie 0 ou 4.

■ 147.

a) Écrire une fonction `somme_cumul` qui prend en entrée une liste `L` d'entiers et qui renvoie une liste `M` des sommes cumulées, c'est à dire une liste `M`, de même longueur que `L`, telle que $M[i] = \sum_{k=0}^{i} L[k]$ pour tout indice i de `L`.

b) On propose à deux joueurs A et F la résolution d'une suite infinie de problèmes numérotés $0,1,\dots,n,\dots$
Ils débutent la résolution à l'instant 0 du problème numéro 0. Dès que l'un des joueurs termine la résolution du problème k il passe au problème $k+1$.
Le numéro d'un problème est attribué au joueur qui le résout en premier. En cas de simultanéité de la résolution d'un problème par les deux joueurs celui-ci n'est pas attribué.
On souhaite écrire une fonction Python `repartition(dureesA,dureesF)` qui, étant donnés les deux listes des durées de résolution des n premiers problèmes par les deux joueurs, renvoie les listes des numéros des problèmes attribués à A et F pour les n premiers problèmes.

1) `repartition([7,1,2,1,2],[4,2,4,3,1])` renvoie `([3,4],[0,1])`.
Quel couple renvoie l'exécution de `repartition([3,2,5,1,1,1],[4,2,2,4,4,2])` ?

2) Écrire la fonction recherchée.

■ 148.

a) Écrire une fonction `gene(n)` qui prend en entrée un entier naturel non nul n et retourne une chaîne de n caractères formée aléatoirement, de façon équiprobable, des caractères `'A','C','G','T'`.

b) Écrire une fonction `nbAC(g)` qui prend en entrée une chaine de caractères formée des caractères `'A','C','G', 'T'` et qui retourne le nombre de fois où la séquence `'AC'` est présente.

Par exemple, `nbAC('GAGCACCCTACTTGGCGCGA')` retournera 2.

■ 149.

On dit qu'un nombre est un palindrome s'il s'écrit de la même manière de gauche à droite ou de droite à gauche.

a) Écrire une fonction `Lchiffres(n)` qui renvoie la liste des chiffres de l'écriture de l'entier n en base décimale. Par exemple `Lchiffres(13042)` renvoie `[1, 3, 0, 4, 2]`.

b) Écrire une fonction `palindrome(n)` qui teste un entier n est un palindrome.

■ 150.

Soit un entier naturel n non nul et une liste `t` de longueur n dont les termes valent 0 ou 1. Le but de cet exercice est de trouver le nombre maximal de 0 contigus dans `t` (c'est-à-dire figurant dans des cases consécutives). Par exemple, le nombre maximal de zéros contigus de la liste `t1` suivante vaut 4 :

i	0	1	2	3	4	5	6	7
t1[i]	0	1	1	1	0	0	0	1

i	8	9	10	11	12	13	14
t1[i]	0	1	1	0	0	0	0

a) Ecrire une fonction `nombreZeros(t,i)`, prenant en paramètres une liste `t`, de longueur `n`, et un indice i compris entre 0 et $n-1$, et renvoyant :

- le nombre de zéros consécutifs dans `t` à partir de l'indice i inclus si $t[i] = 0$,
- 0 sinon.

Par exemple, les appels `nombreZeros(t1,4)`, `nombreZeros(t1,1)` et `nombreZeros(t1,8)` renvoient respectivement les valeurs 3, 0 et 1.

b) En déduire une fonction `nombreZeroxMax(t)`, de paramètre `t`, renvoyant le nombre maximal de zéros contigus d'une liste `t` non vide. On utilisera la fonction `nombreZeros`.

■ 151.

a) On considère qu'un nombre est «*adéquat*» si la somme de ses chiffres est un multiple de 10. Ecrire une fonction nommée `adequat(n)` qui renvoie un booléen indiquant si n est «*adéquat*» ou pas.

b) Ecrire une fonction `modification` d'argument un entier naturel n, qui renvoie un nombre p ayant les mêmes chiffres des dizaines, centaines, etc, que n, et avec un chiffre des unités tel que p soit «*adéquat*». Si n est déjà *adéquat*, on aura $n = p$.

■ 152.

Dans cet exercice, un segment $[a, b]$ ($a \leqslant b$) est représenté par une liste de taille 2 : `[a,b]`.

a) Deux segments sont disjoints si leur intersection est vide. Ecrire une fonction `disjoints(i1,i2)` qui teste si les segments `i1` et `i2` sont disjoints. La fonction renvoie le booléen `True` s'ils sont disjoints, `False` sinon.

b) Une « *liste bien formée* » est une liste de segments qui vérifie les propriétés suivantes :

- les segments sont deux à deux disjoints ;
- les segments sont classés par ordre croissant, en considérant qu'un segment `i1` est plus petit qu'un segment `i2` si le maximum de `i1` est inférieur au minimum de `i2`.

Par exemple, les listes `L1 = [[0,1],[2,5],[3,6]]` et

`L2 = [[2,5],[0,1],[3,6]]` ne sont pas bien formées, tandis que la liste

`L3 = [[0,1],[2,3],[4,6]]` est bien formée.

1) Ecrire une fonction `verifie(L)` qui teste si la liste L de segments est bien formée.

2) Écrire une fonction `appartient(x,L)` qui teste si la valeur `x` est élément d'un des segments de la liste bien formée `L`.

■ 153.

a) Écrire une fonction `carres(n)` qui renvoie la liste ordonnée de tous les carrés parfaits (d'entiers naturels) inférieurs ou égaux à n. Par exemple, `carres(10)` donne $[0, 1, 4, 9]$.

b) Vérifier que tous les entiers inférieurs à 2017 peuvent s'écrire comme la somme de quatre carrés d'entiers.

■ 154.

Une matrice carré d'ordre n est dite «*magique*» si elle contient tous les nombres de 1 à n^2, et si les sommes des nombres de chaque ligne, de chaque colonne et de chaque diagonale sont toutes égales à une même constante s.

a) Déterminer la constante s en fonction de n.

b) Créer une fonction `EstMagique(T)` qui renvoie un booléen indiquant si T est magique ou pas.

Une méthode permettant de construire une matrice magique de taille impaire $n = 2p + 1$ est la suivante :

- On construit une matrice de taille n remplie de zéros. On considère cette matrice comme la représentation sur une période d'une matrice infinie n-périodique en lignes et en colonnes.
- On remplace ensuite les zéros de la matrice successivement avec les entiers de 1 à n^2 comme suit :
  - — on met le 1 dans la case située sous la case centrale de la matrice ;
  - — on place ensuite chaque nombre de 2 à n^2 dans la case située ligne suivante et colonne suivante de celles où on a mis le nombre précédent. Si cette case est déjà remplie, on avance encore d'une ligne et on recule d'une colonne.

On admet que la matrice ainsi construite est magique.

3. Écrire une fonction `Magique(P)` qui renvoie la matrice magique de taille $2p + 1$ créée à l'aide de la méthode précédente.

■ 155.

Soit n un entier naturel impair et non multiple de 5. On admet qu'il possède un multiple noté N qui s'écrit en base 10 avec seulement le chiffre 1 (de la forme $111\ldots111$). Le but de cet exercice est de trouver pour un n vérifiant ces conditions le plus petit multiple N de cette forme, et de déterminer le nombre de 1 nécessaires.

a) Ecrire une fonction `QueDesUns` d'un argument n qui renvoie le premier multiple N de n ne s'écrivant qu'avec des 1.

b) Ecrire une fonction `longueur(k)`, donnant le nombre de chiffres de l'entier naturel k.
Indication : on pourra compter le nombre de divisions entières par 10 que l'on peut faire pour obtenir finalement 0.

■ 156. *Suite de Syracuse ou de Collatz*

Soit la suite $(u_n)_{n\in\mathbf{N}}$ définie par $u_0 = N$ (entier naturel non nul) et

$$\forall n \in \mathbf{N},\ u_{n+1} = \begin{cases} u_n/2 \text{ si } u_n \text{ est pair} \\ 3u_n + 1 \text{ si } u_n \text{ est impair} \end{cases}.$$

On conjecture que pour tout entier naturel N non nul, il existe un plus petit entier n tel que $u_n = 1$.
Cet entier est appelé «*durée de vol*» de la suite (u_n) de valeur initiale N.

a) Écrire une fonction `vol(N)` d'argument un entier N, renvoyant la liste de tous les termes de la suite (u_n) de valeur initiale N jusqu'à la première apparition de la valeur 1, ainsi que la durée de vol et la valeur maximale de la suite.

b) Représenter la durée de vol en fonction du point de départ N, pour les valeurs de N inférieures à 1000.

■ 157.

Pour un entier naturel $n \geqslant 2$, on appelle *diviseurs propres* de n les entiers naturels strictement inférieurs à n qui divisent n.
Par exemple, la liste des diviseurs propres de 100 est `[1,2,4,5,10,20,25,50]`.
On va s'intéresser à la somme de ces diviseurs propres. Pour 100, elle vaut par exemple 117.

a) Ecrire une fonction `LDP` d'argument un entier naturel n, qui renvoie la liste de ses diviseurs propres.

b) On dit qu'un entier naturel est *parfait* s'il est égal à la somme de ses diviseurs propres. Ecrire une fonction `parfait(N)` qui renvoie la liste des entiers p parfaits inférieurs ou égaux à N,.

c) On dit que deux entiers sont *amicaux* si chacun est égal à la somme des diviseurs propres de l'autre. Ecrire une fonction `amicaux(N)` qui renvoie la liste de tous les couples (p,q) d'entiers amicaux tels que $p < q \leqslant N$.

■ 158.

On s'intéresse aux nombres de Flavius Josèphe, construits de la sorte :

— on prend la liste des entiers naturels non nuls : 1 2 3 4 5 6 7 8 9 ...

— on enlève les nombres de deux en deux : 1 3 5 7 9 ...

— on enlève les nombres de trois en trois : 1 3 7 9 ...

et ainsi de suite indéfiniment. Les entiers restants sont les nombres cherchés.

a) Ecrire une fonction `enlever(L,i)` qui retire à la liste L tous les éléments dont la position dans L est un multiple de i.
Par exemple, `enlever([1,3,5,7,9,11,13], 3)` doit donner `[1,3,7,9,13]`.

b) Ecrire une fonction `FJ(n)` qui renvoie la liste des nombres de Flavius Josèphe inférieurs ou égaux à n.

■ 159.

Soit N un entier naturel non nul. On cherche à trier une liste L constituée d'entiers naturels strictement inférieurs à N.

a) Écrire une fonction `comptage(L,N)` qui renvoie une liste P dont le k-ième élément désigne le nombre d'occurences de l'entier k dans la liste L.

b) Utiliser la liste P pour en déduire une fonction `tri(L,N)` qui renvoie la liste L triée dans l'ordre croissant.

■ 160.

a) Écrire une fonction `carres(n)` qui renvoie la liste ordonnée de tous les carrés parfaits (d'entiers naturels) inférieurs ou égaux à n. Par exemple, `C1(10)` donne $[0,1,4,9]$.

b) Écrire une fonction `Scarres(n)` qui renvoie la liste ordonnée de tous les entiers naturels inférieurs ou égaux à n qui s'écrivent comme somme de deux carrés d'entiers naturels.

■ 161.

L'objet de cet exercice est d'étudier numériquement la série $\sum \frac{(-1)^{n+1}}{n}$. Pour tout entier naturel n non nul, on note S_n la somme partielle d'ordre n.

a) On admet que cette série est convergente car les suites extraites $(S_{2n})_{n\geqslant 1}$ et $(S_{2n+1})_{n\geqslant 0}$ sont adjacentes. Déterminer une valeur approchée à 10^{-6} près de la somme de cette série.

b) On modifie l'ordre des termes de la série en prenant alternativement 2 termes positifs et 3 termes négatifs comme suit :

$$\left(1+\frac{1}{3}\right)+\left(-\frac{1}{2}-\frac{1}{4}-\frac{1}{6}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+\left(-\frac{1}{8}-\frac{1}{10}-\frac{1}{12}\right)+\cdots$$

Ecrire une fonction `SPD(n)` d'argument un entier naturel n non nul et qui renvoie la liste des n premières sommes partielles de cette série réordonnée.

■ 162.

Pour un entier naturel n non nul fixé, on appelle *vecteur creux* de $\mathbf{R}^n$ un vecteur dont au moins la moitié des coefficients sont nuls. On code ces vecteurs à l'aide d'une liste de deux listes. La première est la liste des valeurs des coefficients non nuls, la seconde celle de leurs indices, rangés en ordre croissant.
Par exemple, le vecteur $v = (1\ 3\ 0\ 4\ 0\ 0\ 1\ 2\ 0\ 0\ 0\ 0)$ est codé par `[[1,3,4,1,2],[0,1,3,6,7]]`.
On a en effet $v_0 = 1$, $v_1 = 3$, $v_3 = 4$, $v_6 = 1$ et $v_7 = 2$. Les autres coefficients sont nuls.

a) Écrire une fonction `creux(v)` qui renvoie un booléen indiquant si v est un vecteur creux ou pas.

b) Écrire une fonction `coder(v)` qui renvoie le codage « creux » du veecteur `v`.

c) Écrire une fonction `decoder(C,n)` qui restitue le vecteur de longueur n à partir de son codage « creux » C.

■ 163.

Soit `n` un entier vérifiant `n` $\leqslant 26$. On souhaite écrire un programme qui code un mot en décalant chaque lettre de l'alphabet de `n` lettres.
Par exemple, pour $n = 3$, le décalage sera le suivant :

avant décalage	a	b	c	d	…	…	x	y	z
après décalage	d	e	f	g	…	…	a	b	c

Le mot `oralensam` devient ainsi `rudohqvdp`.
On considère la chaîne de caractère `Alph = "abcdefghijklmnopqrstuvwxyz"`.

a) Écrire une fonction `decalage(n,lettre)` qui renvoie la lettre obtenue après un décalage de n lettres à partir de la lettre `lettre`.

b) Écrire une fonction `codage(n,mot)` qui renvoie le codage de la chaîne de caractère `mot` obtenu avec un décalage de `n` lettres.

c) Comment peut-on décoder un mot codé ?

■ 164.

On considère des triangles de nombres de la forme

```
1 1 0 0 1 0 1
0 1 0 1 1 1
1 1 1 0 0
0 0 1 0
0 1 1
1 0
1
```

Chaque ligne est constituée de 0 et de 1. La première est donnée et les suivantes sont construites de la manière suivante : on prend les deux entiers situés respectivement au-dessus et au-dessus à droite de son emplacement. On les additionne et on garde le reste de la division euclidienne de cette somme par 2.
On représente le triangle comme une liste de listes.
Un triangle de ce type qui compte autant de 0 que de 1 est dit *équilibré*.

a) **1)** Ecrire une fonction `suivante(L)` qui prend en argument une ligne `L` constituée de 0 et 1 et qui renvoie la liste représentant la ligne suivante du triangle.

2) Ecrire une fonction `triangle(L)` qui renvoie la liste de listes représentant le triangle de première ligne `L`.

b) Ecrire une fonction `equilibre(T)` qui teste si le triangle `T` est équilibré.

■ 165.

Un échiquier est un plateau de 8 lignes et 8 colonnes. Ces lignes et ces colonnes seront, dans cet exercice, numérotées de 0 à 7. Une position de l'échiquier est un couple (i, j) d'entiers compris entre 0 et 7 inclus, avec i le numéro de ligne et j le numéro de colonne.
Un cavalier placé sur l'échiquier se déplace en bougeant de 2 cases dans une direction (verticale ou horizontale) et de 1 case perpendiculairement. Si le cavalier est loin des bords de l'échiquier, il a 8 possibilités de déplacements, mais il en a moins s'il est près des bords.

a) Ecrire une fonction `coupsSuivants(i,j)` qui renvoie la liste des positions que peut atteindre un cavalier placé en (i, j) en un seul coup.

b) Ecrire une fonction `cavalier(a,b)` qui renvoie un tableau T carré d'ordre 8 tel que $T[i, j]$ est le nombre minimum de coups nécessaires à un cavalier placé en position (a, b) pour arriver à la position (i, j).

■ 166.

Soit Ω_V le sous-ensemble $\{(1,0),(-1,0),(0,1),(0,-1)\}$ de l'espace vectoriel $\mathbf{R}^2$.
Soit (V_n) une suite de variables aléatoires à valeurs indépendantes suivant la même loi uniforme sur Ω_V.
On définit une suite de variables aléatoires (X_n) en posant :

$$X_0 = (0,0), \text{ et } \forall n \in \mathbf{N},\ X_{n+1} = X_n + V_n$$

On appelle *"trajectoire de longueur n"* la ligne brisée aléatoire reliant les points de coordonnées $X_0, X_1, \ldots, X_n$.

a) Écrire une fonction `trajectoire(n)` qui renvoie le tirage d'une trajectoire de longueur n sous forme d'une liste de couples `[x,y]`.

b) Écrire une fonction `premierRetour(m)` qui renvoie le plus petit entier non nul $n \leqslant m$ tel que $X_n = (0,0)$ s'il existe, et -1 sinon.

■ 167.

On considère la fonction ϕ définie comme suit : $\phi(k)$ est le nombre composé des chiffres décrivant le nombre k. Par exemple :

1112	$\mapsto$	3112	(trois uns, un deux)
29	$\mapsto$	1219	(un deux, un neuf)
333	$\mapsto$	33	(trois trois)
1211	$\mapsto$	111221	(un un, un deux, deux uns)

Dans cet exercice, chaque entier naturel sera codé sous forme de liste de chiffres.

a) Écrire une fonction `occur(L,i)` qui prend en entrée une liste `L` d'entiers, un entier `i` et qui compte le nombre de `L[i]` consécutifs dans `L` à partir de la position `i`. Par exemple, si `L=[1,2,1,1,1,2]`, `occur(L,0)` renvoie 1 tandis que `occur(L,2)` renvoie 3, et `occur(L,-1)` renvoie 1.

b) Écrire une fonction `phi(L)` d'argument une liste `L` de chiffres codant un nombre k, qui renvoie la liste des chiffres de $\phi(k)$. Par exemple, `phi([1,2,1,1])` doit donner `[1,1,1,2,2,1]`.

■ 168.

Pour un entier naturel n non nul fixé, on appelle *vecteur creux* de $\mathbf{R}^n$ un vecteur dont au moins la moitié des coefficients sont nuls. On code ces vecteurs à l'aide d'une liste de deux listes. La première est la liste des valeurs des coefficients non nuls, la seconde celle de leurs indices, rangés en ordre croissant.
Par exemple, le vecteur $v = (1,3,0,4,0,0,1,2,0,0,0,0)$ est codé par `[[1,3,4,1,2],[0,1,3,6,7]]`.
On a en effet $v_0 = 1$, $v_1 = 3$, $v_3 = 4$, $v_6 = 1$ et $v_7 = 2$. Les autres coefficients sont nuls.

a) Écrire une fonction `coder(v)` qui renvoie le codage « creux » du veecteur `v`.

b) Ecrire une fonction `pscal(C1,C2)` de deux arguments `C1` et `C2`, codages creux de deux vecteurs v_1 et v_2, qui renvoie le produit scalaire de ces deux vecteurs.

■ 169.

Une méthode pour estimer numériquement l'aire du disque de centre O et de rayon 1 est la suivante : on tire au hasard N points de coordonnées $(x,y) \in [-1,1] \times [-1,1]$; parmi ces points, on compte le nombre n de ceux qui appartiennent au disque. On admet que $\frac{n}{N}$ est une approximation du rapport de l'aire du disque par l'aire du carré.

a) **1)** Écrire une fonction `couple()` qui renvoit un couple de coordonnées d'un point dans $[-1,1] \times [-1,1]$.

2) Écrire une fonction `estim(N)` d'argument N qui renvoie une valeur approchée de π selon le procédé indiqué ci-dessus.

b) Écrire une fonction `nombreN(eps)` qui renvoie le nombre N minimal de point permettant d'obtenir une valeur approchée de π à `eps` près.

■ 170.

On appelle *suite aléatoire binaire* toute suite infinie constituée de tirages de Bernoulli de paramètre $\frac{1}{2}$. L'instant d'apparition d'une séquence dans la suite est le nombre d'éléments de la suite nécessaire à l'apparition complète de la séquence. Par exemple, l'instant d'apparition de $[1,0,1,1]$ vaut 7 dans la suite $0,1,0,1,0,1,1,\ldots$.

a) Définir une fonction `T(Seq)` qui prend en argument une liste `Seq` composée de 0 et de 1 et qui renvoie l'instant d'apparition de `Seq` dans une suite aléatoire binaire.

b) Estimer numériquement l'instant d'apparition moyen. de $[1,0,1,1]$.

■ 171.

On cherche à payer une somme n (entier naturel) avec des pièces de 1, 2 et 5. On veut obtenir exactement n.

a) Définir une fonction `P5(n)` qui donne le nombre de façons de payer la somme `n` avec des pièces de valeur 5. La fonction renvoie donc 1 si `n` est un multiple de 5 et 0 sinon.

b) Définir une fonction `P25(n)` qui donne le nombre de façons de payer la somme `n` avec des pièces de valeur 2 ou 5.

c) Définir une fonction `P125(n)` qui donne le nombre de façons de payer la somme `n` avec des pièces de valeur 1, 2 ou 5.

■ 172.

On dit que que triplet d'entiers (a,b,c) est « pythagoricien » si $1 \leqslant a \leqslant b \leqslant c$ et $a^2 + b^2 = c^2$.

a) Écrire une fonction `couple(k)` qui renvoie tous les couples d'entiers (a,b) tels que $1 \leqslant a \leqslant b \leqslant k$.
Par exemple, couple(3) renvoie
`[(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)]`.

b) Écrire pythagoricien(k) qui renvoie la liste de tous les triplets pythagoriciens constitués d'entiers inférieurs à k.

■ 173.

On donne une liste L de longueur multiple de 3 d'éléments deux à deux distincts.

a) Ecrire une fonction `alterner(L)` qui découpe L en 3 listes $L1, L2, L3$ de longueurs égales $L = L_1 + L_2 + L_3$ et renvoie la liste M construite en prenant successivement les éléments de L_3, L_2 et L_1.
Vérifier que `alterner([1,2,3,4,5,6])` renvoie bien `[5,3,1,6,4,2]`.

b) Écrire une fonction `retour(L)` qui renvoie le nombre de fois qu'il faut appliquer `alterner` à une liste L de longueur $3n$ pour que le premier élément de L se retrouve à nouveau en première position.

■ 174.

a) Créer une fonction qui renvoie le maximum d'une liste non vide sans utiliser les fonctions min ou max.

b) Créer une fonction qui renvoie les deux plus grands éléments d'une liste qui contient au moins deux éléments distincts.

■ 175.

Étant donné un ensemble E et un sous-ensemble A de E tout deux représentés par des listes, on peut coder A à l'aide d'une liste C de la manière suivante : pour `E = [5,1,3,2]` et `A = [2,5]`, on donne `C = [1,0,0,1]` avec `C[k] = 1` quand `E[k]` appartient à A, et 0 sinon.

a) **1)** Ecrire une fonction `coder(E,A)` qui prend en argument un ensemble renvoie la liste C.

2) Ecrire une fonction `decoder(E,C)` qui renvoie la liste A.

b) Ecrire une fonction `inter(C1,C2)` prenant en argument les listes $C1$ et $C2$ correspondant à deux sous-ensembles $A1$ et $A2$ de E et qui teste si l'intersection de $A1$ et $A2$ est vide.

■ 176.

On considère deux urnes numérotées 0 et 1. Initialement, l'urne 1 contient N particules et l'urne 0 est vide.
On prélève une particule et on la place dans l'urne qui ne la contenait pas. Chaque particule ayant la même probabilité d'être prélevée, on répète l'expérience n fois.
On note X_k la variable aléatoire égale au nombre de particules dans l'urne 0 à l'issue du $k^{\text{ème}}$ tirage.

a) Créer une fonction `suivant(x,N)` qui prend en argument le nombre x de boule dans l'urne 0 à un instant donné et N, et qui renvoie le nombre de boule dans l'urne 0 après le tirage d'une boule.

b) Créer une fonction `simulation(n,N)` qui renvoie la liste $[X_0, \ldots, X_n]$ pour un nombre n de tirages avec N particules.

■ 177.

On considère une liste constituée d'entiers naturels non nuls. Un nombre est dit « absolument majoritaire » dans cette liste quand son nombre d'occurences dépasse la moitié de la longueur de la liste.

a) Créer une fonction `compter(L,x)` qui renvoie le nombre d'occurences du nombre x dans la liste L.

b) Écrire une fonction `majo(L)` qui renvoie le nombre absolument majoritaire de L si elle en possède un, 0 sinon.

c) On admet que si un nombre est absolument majoritaire dans une liste, alors il l'est forcément dans une des deux listes qui séparent la liste du début en deux.
En déduire une fonction récursive `majo2(L)` qui renvoie le nombre absolument majoritaire de L si elle en possède un, 0 sinon.

■ 178.

On se place sur une droite horizontale graduée, et on se donne deux pions initialement en 0.
On lance un dé à 6 faces pour déplacer vers la droite le pion situé à la graduation inférieure.

a) Ecrire une fonction `deplacement(a,b)` qui prend pour argument les positions a et b deux deux pions, et qui renvoie les nouvelles positions après un lancer.

b) Ecrire une fonction `arrivee` sans argument qui renvoie la première graduation où les deux pions se rencontrent à nouveau.

■ 179.

a) Écrire une fonction `coupures(L)` qui prend en entrée une liste `L`, et qui renvoie la liste des indices k tels que la k-ième valeur soit différente de la $k-1$-ième, en lui adjoignant en fin la taille de la liste.
Exemple : `coupures([1,1,0,0,0,1,1,1,0,1,1])` renvoie `[2,5,8,9,11]`.

b) Écrire une fonction `CP(L)` qui prend en entrée une liste `L`, et qui renvoie les listes constituées respectivement des valeurs qui composent `L` et du nombre de répétitions de ces valeurs.
Exemples : CP([1,1,0,0,0,1,1,1,0,1,1]) renvoie [1,0,1,0,1],[2,3,3,1,2].

■ 180.

On appelle décomposition d'un entier naturel n en somme de p entiers, toute liste $L = [a_0, a_1, \dots a_{p-1}]$ strictement croissante d'entiers telle que $a_0 + \cdots + a_{p-1} = n$.

a) Ecrire une fonction `dcp2(n)` qui renvoie la liste de ses décompositions en somme de deux entiers.
Par exemple, dcp2(12) renverra $[[2,10],[3,9],[4,8],[5,7]]$

b) En déduire une fonction `dcp3(n)` qui renvoie la liste de ses décompositions en somme de 3 entiers.

■ 181.

On considère des listes de taille n non nul constituées d'éléments deux à deux distincts.

a) Écrire une fonction `permutation(L,M)` qui teste si deux listes L et M sont constituées des mêmes éléments à permutation près.

b) Deux listes L et M sont dites équivalentes s'il existe un indice i tel que :
`L = [l0, ..., li, li+1, ..., ln]` et `M = [li+1, ..., ln, l0, ..., li]`.
Écrire une fonction `equiv(L,M)` qui teste si deux listes L et M sont équivalentes.

■ 182.

n joueurs participent à un jeu. Ils tirent une pièce non truquée. Un joueur gagne une manche s'il est le seul à obtenir pile ou face (tous les joueurs font pile et lui face, ou le contraire). Sinon, on relance la pièce.

a) Écrire une fonction `lancer(n)` qui renvoie une liste des résultats des n lancers (1 pour face et 0 pour pile).

b) Écrire une fonction `jeu(n)` qui renvoie le nombre de fois qu'il faut lancer les dés pour terminer la partie.

c) Écrire une fonction `moyenne(n)` qui calcule le nombre moyen de lancers pour une partie à n joueurs.

■ 183.

On se donne $n+1$ boîtes numérotées de 0 à n, et on place une bille dans la première boîte.
A chaque instant, elle a une chance sur deux de rester dans sa boîte, une chance sur deux d'aller dans la suivante.

a) Écrire une fonction `tirer(n)` qui renvoie le numéro de la boîte dans laquelle se trouve la bille au bout de n étapes.

b) Écrire une fonction `repartition(N,n)` qui renvoie la proportion de billes dans chaque boîte lorsqu'on répète l'expérience pour N billes.

■ 184.

On appelle mot une chaîne de caractères composée de 0 et de 1, et ne contenant pas la chaîne '101'.
Par exemple, '1001' est un mot, mais '1101' n'en est pas un.

a) Écrire une fonction `est_mot(c)` qui teste si la chaîne de caractère `c` est un mot.

b) Écrire une fonction `mots(n)` qui renvoie la liste de tous les mots de longueur n. On pourra partir de la liste des mots de longueur 1 et chercher successivement la liste des mots de longueur 2 par ajout d'un 0 ou d'un 1, etc.