Exercice 1

Une fourmi se déplace sur un axe gradué de la manière suivante : à l'instant $n = 0$, elle est en 0 puis, à chaque instant, elle se déplace de $+1$ avec la probabilité p ($p \in]0,1[$) et -1 avec la probabilité $q = 1 - p$.

1. Écrire une fonction Python `position(n,p)` qui prend en entrée un entier naturel n et un réel p, simule cette marche aléatoire et renvoie la position de la fourmi après n déplacements.
2. Écrire une fonction Python `retour_0(n,p)` qui prend en entrée un entier naturel n et un réel p, qui renvoie le nombre de déplacements effectués pour retourner en 0 au cours des n premiers déplacements si cele est arrivé, -1 sinon.

Exercice 2

Les diviseurs propres d'un entier naturel n sont les entiers naturels qui lui sont strictement inférieurs et le divisent. Par exemple, pour $n = 100$, les diviseurs propres sont 1, 2, 4, 5, 10, 20, 25 et 50.

1. Écrire une fonction Python `ldp(n)` prenant en entrée un entier naturel n et renvoyant la liste des diviseurs propres de n.
2. Écrire une fonction Python `sdp(n)` prenant en entrée un entier naturel n et renvoyant la somme des diviseurs propres de n.
3. Un nombre parfait est un entier naturel n égal à la somme de ses diviseurs propres.
   Écrire une fonction Python `parfaits(N)` prenant en entrée un entier naturel N et renvoyant la liste des nombres parfaits n inférieurs ou égaux à N.
4. Deux nombres sont amicaux lorsque l'un est égal à la somme des diviseurs propres de l'autre et inversement.
   Écrire une fonction Python `amicaux(N)` prenant en entrée un entier naturel N et renvoyant la liste des couples de nombres amicaux (p,q) tels que $p < q \leqslant N$.
5. Représenter sous Python le nuage de points des nombres amicaux.

Exercice 3

On considère les suites $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 \in \mathbb{R}$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \dfrac{u_n^2}{n+1}$.

1. Écrire une fonction Python `u(n,a)` qui prend en entrée un entier n et un réel a qui renvoie le terme u_n avec $u_0 = a$.
2. Représenter les 30 premiers termes la suite $(u_n)_{n\in\mathbb{N}}$ pour $a = 1.6616$ puis les 18 premiers termes pour $a = 1.6617$.
   Commenter.

Exercice 4

Pour tout $p \in \mathbb{N}$, on définit sur $\mathbb{R}^+$ la fonction f_p par : $\forall x \in \mathbb{R}^+$, $f_p(x) = \int_0^1 \sqrt{x + t^p}\ \mathrm{d}t$.

1. Utiliser la méthode des rectangles pour définir sous Python la fonction f_p.
2. Écrire une fonction Python `suitef1(p)` qui prend en argument un entier naturel p et renvoie la liste $[f_0(1), \cdots, f_p(1)]$.
3. Représenter sous Python la suite $(f_p(1))_{p\in[\![0,30]\!]}$. Quelle conjecture peut-on faire ?
4. Représenter sous Python la fonction f_p sur l'intervalle $[0,5]$ pour $p \in \{1,2,3,4,5,6\}$. Commenter.
5. Pour un x fixé dans $\mathbb{R}^+$, utiliser Python pour conjecturer la valeur éventuelle de $\lim\limits_{p\to+\infty} f_p(x)$.

Exercice 5

Lors d'un tournoi de foot, 32 équipes sont en liste. Chacune des équipes a un niveau compris entre 1 (très faible) et 5 (très forte). Quand l'équipe n°i de niveau n_i rencontre l'équipe n°j de niveau n_j, la probabilité que l'équipe n°i gagne est égale à $p_{ij} = \dfrac{n_i}{2(n_i + n_j)}$.

1. Construire la liste E contituée des 32 couples(i, n_i) où $i \in [\![0,31]\!]$ et n_i est un entier choisi au hasard entre 1 et 5.

2. On considère (abusivement) que si deux équipes de même niveau se rencontrent alors le match est nul.
   Écrire une fonction Python `match`(i,j) prenant en entrée deux entiers i et j, simulant un match entre les équipes d'indices i et j et renvoyant le numéro de l'équipe gagnante ou le couple (i,j) si le match est nul.
3. On fait 8 groupes de 4 équipes. Construire la liste G constituée de 8 listes, chacune de ces 8 listes étant constituée de 4 équipes choisies au hasard et sans répétition dans E.
4. Écrire une fonction Python `poule`(k) prenant en entrée un entier $k \in [\![0,7]\!]$ simulant tous les matchs du goupe d'indice k de G et renvoyant la liste des 4 couples (i,p_i) où i est l'indice d'une équipe et p_i le nombre de points obtenus pas cette équipe (3 points pour une victoire, 1 point pour un match nul, 0 pour une défaite).
5. Écrire une fonction Python `affiche_scores`() qui affiche le résultat de chaque poule.

Exercice 6

On appelle grille un tableau carré de taille 3×3 rempli avec les entiers de 1 à 9, chaque entier apparaissant une et une seule fois. On dit qu'une grille est un carré magique si les sommes sur chaque ligne, sur chaque colonne, sur chaque diagonale sont égales à une constante.

1. Écrire une fonction Python `grille`() qui génère aléatoirement une grille.
2. Écrire une fonction Python `estmagique`(G) qui prend en entrée une grille et qui renvoie `True` si la grille est un carré magique et `False` sinon.
   Tester cette fonction sur des grilles construites à l'aide de la fonction précédente.
3. Écrire une fonction Python `carremagique`() qui renvoie un carré magique.

Exercice 7

On dispose d'un jeu de 52 cartes (quatre couleurs : pique, coeur, trèfle, carreau ; dans chaque couleur, 13 cartes : As, 2,3,⋯,10,Valet,Dame,Roi).

1. Une carte peut être modélisée par un couple (couleur,valeur). Construire en Python une liste *jeu* modélisant le jeu de 52 cartes.
2. Écrire une fonction Python `testAs(`*carte*`)` qui prend en entrée une carte, qui renvoie `True` s'il s'agit d'un as et `False` sinon.
3. Écrire une fonction Python `main()` qui renvoie une main de 5 cartes choisies au hasard dans le jeu.
4. Écrire une fonction Python `couleur(`m`)` qui prend en entrée une main (5 cartes du jeu), qui renvoie `True` s'il y a une couleur (5 cartes de même couleur) dans cette main et `False` sinon.
5. Écrire une fonction Python `carre(`m`)` qui prend en entrée une main (5 cartes du jeu), qui renvoie `True` s'il y a un carré (4 cartes de même valeur) dans cette main et `False` sinon.
6. Écrire une fonction Python `full(`m`)` qui prend en entrée une main (5 cartes du jeu), qui renvoie `True` s'il y a un full (3 cartes de même valeur et 2 cartes d'une même autre valeur) dans cette main et `False` sinon.

Exercice 8

Dans l'ADN, il existe 4 nucléotides A, T, C et G. Un codon est un triplet de nucléotides. Un séquence d'ADN est une suite de codons terminée par l'un des 3 "codons stop" qui sont TAG, TAA ou TGA. On modélisera un codon par une chaîne de caractère de longueur 3.

1. Construire en Python la `liste_codons` des 64 codons différents.
2. Écrire une fonction Python `sequence(`n`)` qui prend en argument un entier n et renvoie une séquence ADN de longeur $3n$.
3. Écrire une fonction Python `brin(`s`)` qui prend en argument une séquence d'ADN (c'est à dire une chaîne de caractères), renvoie le premier brin d'ADN qui se termine par un "codon stop" s'il y en a un et -1 sinon.
4. Les nucléotides complémentaires sont A-T et C-G.
   Écrire une fonction Python `complementaire(`$b1, b2$`)` qui prend en argument deux brins qui renvoie `True` si ces deux brins sont complémentaires et `False` sinon.
5. Le phénomène de croisement entre deux brins intervient quand les deux brins se brisent au même niveau pour ensuite échanger les fragments résultants.
   Écrire une fonction Python `crossover(`$b1, b2, k$`)` qui prend en argument deux brins et un entier k, qui effectue l'enjambement au niveau k et renvoie les deux nouveaux brins.

Exercice 9

On considère une série statistique X dont les valeurs sont des entiers naturels $x_1, \cdots, x_n$.

1. Écrire une fonction Python `modalites_eff(`X`)` d'argument une liste d'entiers X et renvoie la liste des modalités effectives de cette série rangées dans l'ordre croissant ainsi que la liste des effectifs correspondants.
2. Écrire une fonction Python `mode(`X`)` d'argument une liste d'entiers X et renvoie le mode de cette série, c'est-à-dire la ou les valeurs de la série avec le plus grand effectif.
3. Écrire une fonction Python `moyenne(`X`)` d'argument une liste d'entiers X et renvoie leur moyenne.
4. Écrire une fonction Python `mediane(`X`)` d'argument une liste d'entiers X et renvoie leur médiane.

Exercice 10

Des musées sont composés de collections permanentes P et de collections de réserve R. Dans ces musées, il y a 7 salles et 3 types d'oeuvres (peinture, sculpture, photo). Chaque oeuvre est caractérisée par un triplet (c, s, t) où $c = {}'P'$ ou ${}'R'$, $s \in |[1, 7]|$ est le numéro de la salle et t le type de l'oeuvre. Un musée est défini comme une liste d'oeuvres.

Par exemple $M = [({}'P', 4, 1), ({}'R', 6, 3), ({}'P', 6, 2)]$ est un musée qui contient trois oeuvres : une peinture de la collection permanente dans la salle 4, une photo de la collection de réserve dans la salle 6 et une sculpture de la collection permanente dans la salle 6.

1. Écrire une fonction Python `liste_Musee`(n) qui prend en entrée un entier strictement positif n, qui renvoie une liste de n musées, chacun de ces n musées contient un nombre aléatoire compris entre 1 et 100 d'oeuvres, de types choisis aléatoirement, réparties au hasard dans les 7 salles.
2. Écrire une fonction Python `reserve`(M) qui prend en argument une liste représentant un musée et renvoie le nombre d'oeuvres de la réserve de ce musée.
3. Écrire une fonction Python `eff_salle`(M) qui prend en argument une liste représentant un musée et renvoie la liste des nombres d'oeuvres par salle. Sur l'exemple précédent, la fonction doit renvoyer la liste $[0, 0, 0, 1, 0, 2, 0]$.
4. Écrire une fonction Python `classe`(L) qui prend en argument une liste de musées et renvoie la liste des musées ordonnées suivant le nombre d'oeuvres croissant dans ces musées.

Exercice 11

Le but du crible d'Ératosthène est de déterminer tous les nombres premiers compris entre 1 et un entier n. Partant de la liste des entiers de 1 à n, on retire tous les multiples de 2, puis tous les multiples de 3 etc...

1. Écrire une fonction Python `crible`(L, k) qui prend en entrée une liste L d'entiers stricement positifs et un entier strictement positif k, qui renvoie la liste L privée de tous les multiples de k qu'elle contient.
2. Écrire une fonction Python `Eratosthene`(n) qui prend en entrée un entier strictement positif n et renvoie la liste de tous les entiers premiers inférieurs ou égaux à n.
3. Écrire une fonction Python `test_premier`(n) qui prend en entrée un entier strictement positif n et renvoie `True` si n est premier, `False` sinon.

Exercice 12

Approximation de π par la méthode de Monte-Carlo.

Soit X et Y deux variables aléatoires indépendantes de loi uniforme sur $[0, 1]$.

1. Écrire une fonction Python `Monte_Carlo`(N) qui prend en entrée un entier N et qui permet d'estimer la probabilité de l'événement $[X^2 + Y^2 \leqslant 1]$ pour N réalisations de X et Y.
2. En déduire une valeur approchée de π.

Exercice 13

1. Rappeler la définition d'une loi de Poisson.

   On note $F(\lambda, k)$ la fonction de répartition d'une loi de Poisson de paramètre λ évaluée en $k \in \mathbb{N}$.

   Programmer cette fonction en Python.
2. On définit la fonction $G : \mathbb{R}_+^* \times [0, 1[\to \mathbb{R}$ comme suit : $G(\lambda, x) = \min\{k \in \mathbb{N} \mid F(\lambda, k) > x\}$.

   On admet que la fonction G est bien définie.

   Programmer G en Python. Que simule cette fonction ?

Exercice 14

On considère un plateau de n cases ($n > 0$). Sur chaque case se trouve un numéro de case (un même numéro peut être inscrit sur plusieurs cases différentes).

Le jeu consiste à placer au départ un pion sur la première case (d'indice 0). Ensuite, à chaque étape, on déplace le pion sur la case d'indice le numéro marqué sur la case sur laquelle se trouve le pion. Par exemple, sur le plateau suivant

2	4	1	3	0

, $n = 5$. La première étape du jeu consiste à placer le pion sur la case d'indice 0 où se trouve le numéro 2, on place ensuite le pion sur la case d'indice 2 où se trouve le numéro 1 puis sur la case d'indice 1 où se trouve le numéro 4 etc.

Sous Python, on représente un plateau par une liste. Pour l'exemple, la liste $[2,4,1,3,0]$.

1. Écrire une fonction Python *plateau*(n) qui prend en argument un entier strictement positif n et renvoie un plateau de jeu.

2. On appelle point fixe du plateau une case sur laquelle est inscrit son propre numéro. Sur l'exemple, la case 3 est un point fixe (il n'y en a pas d'autres).

Écrire une fonction Python *compter*(*plat*) qui prend en argument une liste *plat* représentant un plateau et renvoie le nombre de points fixes de ce plateau.

3. Écrire une fonction Python *avancer*(*plat*, p) qui prend en argument une liste *plat* représentant un plateau et un entier naturel p et qui renvoie le numéro de la case sur laquelle le pion arrive à l'issue de p étapes de jeu. Par exemple, *avancer*($[2,4,1,3,0], 1$) renvoie 2 et *avancer*($[2,4,1,3,0], 3$) renvoie 4.

Exercice 15

Une grille de *Takuzu* ou *Binairo* est une grille à n lignes et n colonnes où n est un entier pair.

Au départ, chaque case peut contenir 0, 1 ou être vide.

En Python, une grille est codée par une liste de listes, les cases vides étant représentées par `None`.

Par exemple, la grille

0	1		0
1		0	1
0	0	1	1
1	1	0	

est codée par la liste [[0, 1, `None`, 0], [1, `None`, 0, 1], [0, 0, 1, 1], [1, 1, 0, `None`]].

Dans la suite, on identifie la grille et la liste qui la représente.

1. Écrire une fonction Python prenant en argument un entier n et renvoyant une grille de *Takuzu*, chaque case contient 0, 1 ou rien de manière aléatoire.

2. Écrire une fonction Python prenant en argument une grille et renvoyant le nombre de cases non remplies.

3. Écrire une fonction Python prenant en argument une grille et renvoyant un booléen testant s'il existe deux lignes identiques dans la grille.

Exercice 16

Soit $p \in]0,1[$. Une puce peut occuper quatre positions numérotées de 0 à 3.

À chaque instant, la puce se déplace ou non suivant le protocole :

• 0 et 3 sont des *puits* : si la puce s'y trouve, elle y reste définitivement

• si la puce est en 1 à l'instant n alors, à l'instant suivant, elle saute en 0 avec la probabilité p ou en 2 avec la probabilité $q = 1 - p$

• si la puce est en 2 à l'instant n alors, à l'instant suivant, elle saute en 1 avec la probabilité p ou en 3 avec la probabilité $q = 1 - p$

À l'instant 0, la puce se trouve en 1.

1. Écrire une fonction Python *saut*(p, k) prenant en entrée le réel p et un entier k de $\{0,1,2,3\}$ égal à la position de la puce à un intant donnée, renvoyant la position de la puce à l'instant suivant.

2. Écrire une fonction Python *position*(p, n) prenant en entrée le réel p et un entier naturel n, renvoyant la liste des n premières positions de la puce.

Exercice 17

1. Écrire une fonction *strict_croissance*(L) prenant une liste d'entiers en paramètre et renvoyant `True` si la liste est strictement croissante et `False` sinon.
2. Écrire une fonction *strict_monotone*(L) prenant une liste d'entiers en paramètre et renvoyant `True` si la liste est strictement croissante et `False` sinon.

Exercice 18

1. Écrire une fonction Python *nb_vrai*(L) prenant en entrée une listes de booléens et renvoyant le nombre d'occurence de la valeur `True`.

   Par exemple, *nb_vrai*(`[False,True,True]`) doit renvoyer 2.
2. Écrire une fonction Python *bloc_vrai*(L) prenant en entrée une listes de booléens et renvoyant `True` si la valeur `True` est présente dans L et que tous les `True` sont côte-à-côte, dans le même bloc, `False` sinon.

   Par exemple, *bloc_vrai*(`[False,True,True,True,False,False]`) doit envoyer `True` car il y a un unique bloc de `True`

   mais *bloc_vrai*(`[False,True,True,True,False,False,True]`) doit envoyer `False` car il y a deux blocs de `True`.

Exercice 19

1. Écrire une fonction Python *Cesar*(C,n) prenant en argument une chaîne de caractères C constituée exclusivement de lettres de l'alphabet et un entier naturel n renvoyant la chaîne C dont toutes les lettres ont été décalées de n rangs dans l'alphabet. Par exemple,

```
>>> C = 'mathematiques'

>>> cesar(C,10)
'wkdrowkdsaeoc'
```

2. Écrire une fonction Python *pyramide*(C,n) prenant en argument une chaîne de caractères C et un entier naturel strictement positif n, affichant la pyramide de hauteur n construite avec les lettres de la chaîne de caractères. Par exemple,

```
>>> C = 'mathematiques'

>>> pyramide(C,10)
m
at
hem
atiq
uesma
themat
iquesma
thematiq
uesmathem
atiquesmat
```

Exercice 20

1. Un étudiant note chronologiquement toutes ses recettes et toutes ses dépenses les unes à la suite des autres. Il a ainsi une suite de nombres dont certains sont positifs (les recettes) et d'autres sont négatifs (les dépenses). Écrire une fonction Python `budget`(L) prenant en argument une telle liste et renvoyant le total de ses recettes et le total de ses dépenses.
2. On dispose de billets et pièces de 100€, 50€, 20€, 10€, 5€, 2€, 1€, 0.50€, 0.20€, 0.10€, 0.05€, 0.02€ et 0.01€. Écrire une fonction Python `decompose`(S) prenant en argument un réel S et affichant le nombre de chaque billets ou pièces nécessaires. Par exemple :

```
>>> decompose(208.74)
100 euros : 2
5 euros : 1
2 euros : 1
1 euros : 1
0.5 euros : 1
0.2 euros : 1
0.02 euros : 2
```

Exercice 21

On s'intéresse aux nombres de Flavius Josèphe dont la liste est construite de la manière suivante : on prend la liste des entiers strictements positifs 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,$\cdots$, on en supprime un sur deux, il reste 1,3,5,7,9,11,13,15,$\cdots$, on en supprime un sur trois, il reste 1, 7,9,13,15,$\cdots$ et ainsi de suite. Les nombres de Flavius Josèphe sont ceux qui ne sont jamais supprimés

1. Écrire une fonction Python `retire`(L,i) d'arguments L et un entier naturel i, renvoyant la liste issue de L dont les termes de rang un multiple de i ont été supprimés.
2. Écrire une fonction Python `FJ`(n) prenant en entrée un entier strictement positif n et renvoyant la liste des nombres de Flavius Josèphe inférieurs où égaux à n.
3. Écrire une fonction Python `U`(n) prenant en entrée un entier strictement positif n et renvoyant le nombre de nombres de Flavius Josèphe inférieurs où égaux à n.
4. Représenter la suite $\left(\dfrac{4n}{U(n)^2}\right)_{n\in\mathbb{N}^*}$ et estimer la limite de la suite. Donner alors un équivalent de $U(n)$.

Exercice 22

Le code creux consiste à coder un vecteur u sans les 0 qu'il comporte, qui constituent une information inutile pour les calculs. Ce code comporte une première liste des coordonnées non nulles de u ainsi que la liste des indices de ces coordonnées. Par exemple, si $u = (1,2,0,5,8,0,0,0,7)$ alors le code creux de u est $[[1,2,5,8,7],[0,1,3,4,8]]$.

1. Écrire une fonction Python `code_creux`(U) prenant en entrée une liste U constituée des coordonnées d'un vecteur u, renvoyant les listes du code creux de u.
2. Écrire une fonction Python `decode_creux`(L) prenant en entrée une liste de deux listes C et I, renvoyant une liste des coordonnées d'un vecteur u dont le code creux est $[C,I]$.

Exercice 23

On souhaite approcher une fonction f définie et continue sur une intervalle $[a,b]$ par un polynôme. On suppose que $a < b$.

Pour cela, on choisit un entier naturel n et on pose :$\forall k \in |[0,n]|$, $a_k = a + k\dfrac{b-a}{n}$, $L_k = \displaystyle\prod_{\substack{i=0\\ i\neq k}}^{n} \frac{X-a_i}{a_k-a_i}$.

$L = \displaystyle\sum_{k=0}^{n} f(a_k)L_k$. est appelé le polynôme d'interpolation de Lagrange.

1. Écrire une fonction Python `subdivision`(a,b,n) qui prend en entrée les réels a et b et un entier n, qui renvoie la liste des a_k.
2. Écrire une fonction Python `Lagrange`(sub,k) qui prend en entrée une liste $sub = [a_0,\cdots,a_k]$ de réels et qui renvoie la fonction polynôme L_k.
3. Écrire une fonction Python `interpol`(f,a,b,n) prenant entrée une fonction f, des réels a et b et un entier n, renvoyant la fonction polynôme $L = \displaystyle\sum_{k=0}^{n} f(a_k)L_k$
4. Écrire une fonction Python `affiche`(f,a,b,n) prenant entrée une fonction f, des réels a et b et un entier n, qui affiche dans une même fenêtre la courbe représentative de f et celle de L.
5. Tester cette fonction pour $f(x) = \sin x$ sur $[-2\pi,2\pi]$ pour $n \in \{2,3,5,8,10\}$. Commenter.

Exercice 24

Soit f la fonction définie sur $\mathbb{R}_+^*$ par $\forall x \in \mathbb{R}_+^*$, $f(x) = \displaystyle\int_x^{+\infty} \frac{\sin t}{t^2}\,\mathrm{d}t$.

Sous Python, après avoir importé la fonction `quad` du module `scipy.integrate`, l'instruction `quad`(h,a,b) renvoie une valeur approchée de $\displaystyle\int_a^b h(t)\,\mathrm{d}t$ ainsi que l'erreur d'approximation.

1. Représenter sous Python la fonction f sur $]n\pi,(n+1)\pi[$ pour $n \in \{1,2,3,4,5,6\}$.
   Quelle conjecture peut-on faire quant au nombre de solutions de l'équation $f(x) = 0$ sur l'intervalle $]n\pi,(n+1)\pi[$.
2. On suppose que : $\forall \in \mathbb{N}$, $\exists! r_n \in]n\pi,(n+1)\pi[\mid f(r_n) = 0$.
   Écrire une fonction Python `approx`(n) prenant en entrée un entier naturel n et renvoyant une valeur approchée de r_n à 10^{-3} près.

Exercice 25

On considère la suite de polynômes définie par : $P_0 = 1$, $P_1 = 2X$ et $\forall n \in \mathbb{N}$, $P_{n+2} = 2XP_{n+1} - P_n$.

1. En utilisant la récursivité, écrire une fonction Python `P(n,x)` prenant en entrée un entier n et un réel x qui renvoie $P_n(x)$.
2. Écrire une fonction Python `prod_scal(i,j)` prenant en entrée deux entiers i et j et renvoyant l'intégrale $\int_{-1}^{1} \sqrt{1-t^2}\, P_i(t)P_j(t)\, \mathrm{d}t$.

   Pour calculer l'intégrale, on pourra utiliser la méthode des rectangles ou la fonction `quad` du module `scipy.integrate`.
3. Appliquer cette fonction à tous les couples $(i,j) \in |[0,8]|^2$. On pourra faire afficher des arrondis à 3 chiffres après la virgule. Que remarque-t-on ?

Exercice 26

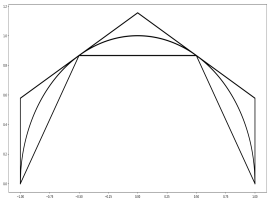
Soit $r \in \mathbb{N}^*$. On considère la grille $G = |[-r,r][^2$, ensemble des points $M(i,j)$ tels que $-r \leqslant i \leqslant r$, $-r \leqslant j \leqslant r$. Une fourmi se déplace dans les quatre directions sur cette grille de la manière suivante : à l'instant $n = 0$, elle est en $(0,0)$ puis, à chaque instant, elle se déplace d'une unité de manière équiprobable dans l'une des directions qui lui sont acessibles.

1. Écrire une fonction Python `deplacement(r,c)` qui prend en argument l'entier r et un couple $c \in G$, qui simule la marche aléatoire et renvoie la position de la fourmi après un déplacement à partir de c.
2. Écrire une fonction Python `position(r,n)` qui prend en argument l'entier r et un entier naturel n, simule la marche aléatoire de la fourmi et renvoie sa position après n déplacements.
3. Écrire une fonction Python `retour_0(r,n)` qui prend en entrée l'entier r et un entier naturel n, qui renvoie le nombre de déplacements effectués pour retourner en $(0,0)$ au cours des n premiers déplacements si cela est arrivé, -1 sinon.

Exercice 27

Approximation de π par la méthode d'Archimède de Syracuse.

Le périmètre d'un demi-cercle de rayon 1 est π. On l'encadre par les périmètres des demi- polygones réguliers à n côtés inscrits et circonscrits à ce demi-cercle :



Pour un polygone régulier à n côtés, l'angle au centre vaut $\frac{2\pi}{n}$. On a donc, pour $n = 6.2^k$, $k \in \mathbb{N}$, le périmètre du demi-cercle inscrit est $p_n = nu_n$ où $u_n = \sin\frac{\pi}{n}$ et le périmètre du demi-cercle circonscrit est $p'_n = nu'_n$ où $u'_n = \tan\frac{\pi}{n}$. On peut définir les suites (u_n) et (u'_n) par : $u_6 = \frac{1}{2}$, $u'_6 = \frac{1}{\sqrt{3}}$ et

$\forall n \in \mathbb{N}^*$, $u_{2n} = \sqrt{\frac{1-\sqrt{1-u_n^2}}{2}}$ et $u'_{2n} = \frac{u_n}{\sqrt{1-u_n^2}}$. On a alors : $\forall k \in \mathbb{N}$, $n = 6.2^k$, $p_n < \pi < p'_n$.

Écrire une fonction Python récursive `u(n)` qui renvoie u_n où $n = 6.2^k$.

En déduire une fonction Python `Archimède(n)` qui renvoie une valeur approchée de π.

Exercice 28

Approximation de π par la formule de Gregory. Soit $x \in]-1,1[$.

La série $\sum(-x^2)^n$ est-elle convergente. Si oui, donner sa somme.

En primitivant, on obtient que la série $\sum \frac{(-1)^n x^{2n+1}}{2n+1}$ est convergente. Quelle est sa somme ?

Écrire une fonction Python `Gregory(n,x)` qui prend en entrée un entier naturel n et un réel x, et renvoie la somme $S_n = \sum_{k=0}^{n} \frac{(-1)^k x^{2k+1}}{2k+1}$. En déduire une valeur approchée de π.

Exercice 29

Dans le plan, on considère les trois points $A(20,20)$, $B(320,20)$ et $C(170,280)$.

On définit la suite de points (P_n) par : P_0 est un point quelconque à l'intérieur du triangle (ABC) (on pourra par exemple choisir $P_0(180,150)$) et $\forall n \in \mathbb{N}$, P_{n+1} est le milieu de l'un des trois segments $[AP_n]$, $[BP_n]$, $[CP_n]$ choisi de manière aléatoire.

Écrire une fonction Python `Spierpinsky(n)` qui prend en entrée un entier naturel n et représente la fractale de Spierpinsky. Tester la fonction pour $n = 30000$.

Exercice 30

Dans le plan, on définit la fractale de Barnsley comme l'ensemble des points $M_n(x_n, y_n)$ défini par : $x_0 = 0.5$, $y_0 = 0$ et $\forall n \in \mathbb{N}$,

$$(x_{n+1}, y_{n+1}) = \begin{cases} (0.5, 0.27y_n) & \text{avec une probabilité de } 2\% \\ (-0.139x_n + 0.263y_n + 0.57, 0.246x_n + 0.224y_n - 0.036) & \text{avec une probabilité de } 15\% \\ (0.17x_n - 0.215y_n + 0.408, 0.222x_n + 0.176y_n + 0.0893) & \text{avec une probabilité de } 13\% \\ (0.781x_n + 0.034y_n + 0.1075, -0.032x_n + 0.739y_n + 0.27) & \text{avec une probabilité de } 70\% \end{cases}$$

Écrire une fonction Python Barnsley(n) qui prend en entrée un entier naturel n et représente la fractale de Barnsley. Tester la fonction pour $n = 30000$.

Exercice 31

Dans le plan, on définit la fractale de Mandelbrot comme l'ensemble des points $M_n(x_n, y_n)$ défini par : $x_0 = 0.5$, $y_0 = 0$ et $\forall n \in \mathbb{N}$,

$$(x_{n+1}, y_{n+1}) = \begin{cases} (0.05x_n, 0.6y_n) & \text{avec une probabilité de } 10\% \\ (0.05x_n, 1 - 0.5y_n) & \text{avec une probabilité de } 10\% \\ (0.46x_n - 0.32y_n, 0.39x_n + 0.38y_n + 0.6) & \text{avec une probabilité de } 20\% \\ (0.47x_n - 0.15y_n, 0.17x_n + 0.42y_n + 1.1) & \text{avec une probabilité de } 20\% \\ (0.43x_n + 0.28y_n, -0.25x_n + 0.45y_n + 1) & \text{avec une probabilité de } 20\% \\ (0.42x_n + 0.26y_n, -0.35x_n + 0.31y_n + 0.7) & \text{avec une probabilité de } 20\% \end{cases}$$

Écrire une fonction Python `Mandelbrot(n)` qui prend en entrée un entier naturel n et représente la fractale de Mandelbrot. Tester la fonction pour $n = 30000$.

Exercice 32

On étudie des séries binaires, composées de 0 et de 1, comme par exemple 0100111011000110 que l'on représente par des listes de 0 ou 1. On cherche à compresser une série bianire dela manière suivante. Le premier terme de la suite compressée est donné par la valeur du début de la suite binaire, puis les termes suivants sont les nombres de valeurs successives égales. Par exemple, la suite binaire 0100111011000110 est représentée par la liste $L = [0,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0]$ et sa suite compressée associée est $[0,1,1,2,3,1,2,3,2,1]$.

1. Écrire une fonction Python `compresse(L)` qui prend en entrée une liste L et renvoie la liste compressée associée.
2. Écrire une fonction Python `decompresse(L)` qui prend en entrée une liste L qui commence par 0 ou 1 et renvoie la liste initiale.
3. Écrire une fonction Python `remplace(L)` qui prend en entrée une liste L correspondant à une suite binaire non compressée et renvoie cette même liste mais dans laquelle les 1 de la plus longue suite de 1 sont remplacés par des 2.

Exercice 33

On étudie des séries binaires, composées de 0 et de 1, comme par exemple 0100111011000110 que l'on représente par la liste $L = [0,1,0,0,1,1,1,0,1,1,0,0,0,1,1,0]$. Les but de cet exercice est de trouver le nombre maximal de 0 consécutifs. Pour l'exemple, ce nombre vaut 3.

1. Écrire une fonction Python `nbre_zeros`(L,i) d'arguments L et un entier $i \leqslant$ longueur de L, qui renvoie : $\begin{cases} \text{le nombre de 0 consécutifs à partir de l'indice } i & \text{si } L[i] = 0 \\ 0 & \text{sinon} \end{cases}$. Pour l'exemple précédent, `nbre_zeros`$(L,3)$ et `nbre_zeros`$(L,10)$ renvoient 1, 2 et 3.

2. En déduire une fonction `nbre_zeros_Max`(L) d'argument L et renvoie le nombre maximal de 0 consécutifs.

Exercice 34

1. Écrire une fonction `carres`(n) d'argument un entier naturel n, qui renvoie la liste ordonnée de tous les carrés inférieurs ou égaux à n. Par exemple, `carres`(10) doit renvoyer $[0,1,4,9]$.

2. Écrire une fonction `Som_Carre`(n) d'argument un entier naturel n, qui renvoie la liste ordonnée de tous les eniters naturels inférieurs ou égaux à n qui s'écrivent comme somme de deux carrés d'entiers.

Exercice 35

Soit $(a,b) \in \mathbb{R}^2$. On représente l'intervalle $[a,b]$ par la liste $[a,b]$.

1. Écrire une fonction `disjoints`$(I1,I2)$ d'arguments deux intervalles, renvoyant `True` s'ils sont disjoints et `False` sinon.

2. On dit qu'une liste de d'intervalles est bien formée si les intervalles qui la constituent sont deux à deux disjoints et classés par ordre croissant (le maximum d'un intervalle est inférieur au minimum de l'intervalle suivant).

a. Écrire une fonction `bien_formee`(L) d'argument une liste d'intervalles, qui renvoie `True` si L est bien formée et `False` sinon.

b. Écrire une fonction `appartient`(x,L) d'arguments un réel x et une liste L bien formée, qui renvoie `True` si x appartient à l'un des intervalles de L et `False` sinon.

Exercice 36

Soit n un entier naturel impair et non multiple de 5. On admet que n admet un multiple N qui s'écrit seulement avec des 1. Par exemple, 3 admet pour multiple 111, 7 admet pour multiple 111111.

1. Écrire une fonction `Que_des_1`(n) d'argument un entier naturel impair et non multiple de 5, qui renvoie le plus petit entier N multiple de n qui s'écrit seulement avec des 1.

2. Écrire une fonction `Nbre_de_1`(n) d'argument un entier naturel impair et non multiple de 5, qui renvoie le nombre de "1" du plus petit entier N multiple de n qui s'écrit seulement avec des 1.

3. Représenter la suite des longueurs des nombres N en fonction de n.

Exercice 37

Soit $N \in \mathbb{N}^*$ et la suite $(u_n)_{n\in\mathbb{N}}$ définie par : $u_0 = N$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$.

On conjecture que, pour tout entier non nul N, il existe un entier n tel que $u_n = 1$. Soit d le plus petit entier tel que $u_d = 1$. Cet entier d est appelé la "durée de vol" de la suite $(u_n)_{n\in\mathbb{N}}$.

1. Écrire une fonction `duree`(N) d'argument un entier strictement positif N qui renvoie la liste $[u_0,\cdots,u_d]$.

2. Représenter la durée de vol en fonction de N pour $N \in |[1,1000]|$.

Exercice 38

Pour tout $n \in \mathbb{N}^*$, on note $S_n = \sum_{k+1}^{n} \frac{(-1)^{k+1}}{k}$.

1. Représenter sous Python les suites extraites des termes de rangs pairs et des termes de rangs impairs $(S_{2n})_{n\in\mathbb{N}^*}$ et $(S_{2n+1})_{n\in\mathbb{N}}$. Quelle conjecture peut-on faire concernant la suite $(S_n)_{n\in\mathbb{N}^*}$?
2. En déduire une fonction Python qui renvoie une valeur approchée de la somme de la série $\sum \frac{(-1)^{n+1}}{n}$ à 10^{-6} près.

Exercice 39

Un échiquier est un plateau de 8 lignes et 8 colonnes. Chaque case est représentée par un couple $(i,j) \in |[0,7]|$.

Un cavalier se déplace en bougeant de trois cases : une dans une directions (horizontale ou verticale) et deux dans l'autre direction.

1. Écrire une fonction `un_coup`(c) d'argument une case c (c'est-à-dire un couple d'entiers) qui renvoie la liste des positions que peut atteindre le cavalier placé en c en un seul coup.
2. Écrire une fonction `cavalier`(c) d'argument une case c (c'est-à-dire un couple d'entiers) qui renvoie un tableau carré d'ordre 8 T tel que, pour tout $(i,j) \in |[0,7]|$, $T[i,j]$ est le nombre minimum de coups nécessaires à un cavalier placé en c pour arriver à la position (i,j). On aura par exemple : si $c = (a,b)$, $T[a,b] = 0$, $T[a+1,b+2] = 1$ si $a+1 \leqslant 7$ et $b+2 \leqslant 7$.

Exercice 40

Étant donnée une liste L constituée de 0 et 1, on contruit un triangle de la manière suivante : en dessous de L, on construit la liste dont chaque terme est le reste de la division euclienne de la somme des deux entiers situés au-dessus et au-dessus à droite de son emplacement par 2. On réitère ce processus pour créer les lignes suivantes.

1. Écrire une fonction `suivante`(L) d'argument une liste L de 0 et 1 et qui renvoie la liste suivante.
   Par exemple pour $L = [1,1,0,0,1,0,1]$, la fonction doit renvoyer la liste $[0,1,0,1,1,1]$.
2. Écrire une fonction `triangle`(L) d'argument une liste L de 0 et 1 et qui renvoie le triangle complet.
3. Un triangle ainsi construit est dit équilibré s'il contient autant de 0 que de 1.
   Écrire une fonction `equilibre`(T) d'argument un triangle T, qui renvoie `True` si le triangle est équilibré et `False` sinon.

Exercice 41

On dit que le triplet d'entiers (a,b,c) est "pythagoricien" si $0 \leqslant a \leqslant b \leqslant c$ et $a^2 + b^2 = c^2$.

1. Écrire une fonction `couple`(n) d'argument un entier naturel strictement positif n et qui renvoie la liste des couples d'entiers (a,b) tels que $1 \leqslant a \leqslant b \leqslant n$.
2. Écrire une fonction `pythagoricien`(n) d'argument un entier naturel strictement positif n et qui renvoie la liste des triplets pythagoriciens (a,b,c) d'entiers inférieurs ou égaux à n.

Exercice 42

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 au hasard et on la change d'urne.

On note X_k la variable aléatoire égale au nombre de particules dans l'urne 0 à l'issue du k-ième prélèvement.

1. Écrire une fonction `suivant`(k,N) d'arguments un entier k égal au nombre de boules dans l'urne 0 à un instant donné et l'entier N, qui renvoie le nombre de boules dans l'urne 0 à l'instant suivant.
2. Écrire une fonction `simulation`(n,N) d'arguments deux entiers n et N qui renvoie la liste $[X_0, \cdots, X_n]$ pour n prélèvements.

Exercice 43

On appelle décomposition d'un entier naturel n en somme de p entiers toute liste strictement croissante de p entiers strictement positifs dont la somme vaut n.

1. Écrire une fonction `decomposition2`(n) qui renvoie la liste des couples (a,b) de deux entiers tels que $0 < a < b$ et $a + b = n$.
2. En déduire une fonction `decomposition3`(n) qui renvoie la liste destriplets (a,b,c) de trois entiers tels que $0 < a < b < c$ et $a + b + c = n$.

Exercice 44

n joueurs jettent une pièce non truquée. Un joueur gagne une manche de ce jeu s'il est le seul à obtenir l'un des côtés Pile ou Face. Le jeu est terminé lorsqu'un joueur a gagné une manche.

1. Écrire une fonction `manche`(n) d'argument un entier n qui renvoie la liste des résultats des lancers des n joueurs.
2. Écrire une fonction `jeu`(n) d'argument un entier n qui renvoie le nombre de manches nécessaires pour que le jeu s'arrête avec n joueurs.
3. Écrire une fonction `moyenne`(n) d'argument un entier n qui renvoie le nombre moyen de manches nécessaires pour que le jeu s'arrête avec n joueurs.

Exercice 45

On considère une liste d'entiers naturels strictement positifs. Un entier de cette liste est dit "absolument majoritaire" quand le nombre de fois où il apparait dans la liste dépasse la moitié de la la longeur de la liste.

1. Écrire une fonction `compter`(L,x) d'arguments une liste d'entiers L et un entier x, qui renvoie le nombre d'occurences de l'entier x dans la liste L.
2. Écrire une fonction `majoritaire`(L) d'arguments une liste d'entiers L qui renvoie l'entier absolument majoritaire de L si elle en possède un, 0 sinon.

Exercice 46

On dispose de $n+1$ boîtes numérotées de 0 à n. À l'intant 0, on place une bille dans la boîte 0 et, à chaque instant, la bille a une chance sur deux de rester dans la boîte où elle est ou bien de passer dans la boîte suivante.

1. Écrire une fonction `numero`(n) d'argument un entier n qui renvoie le numéro de la boîte dans la quelle se trouve la bille à l'instant n.
2. On dispose maintenant de N billes qui sont toutes placées initialement dans la boîte 0. À chaque instant, chacune des billes a une chance sur deux de rester dans la boîte où elle est ou bien de passer dans la boîte suivante. Pour tout $k \in [\![0,n]\!]$, on note X_k le nombre de billes dans la boîte k.

   Écrire une fonction `repartition`(N,n) d'argument deux entiers N et n qui renvoie la liste $[X_0,\cdots,X_n]$.

Exercice 47

On définit la suite de Robinson de la manière suivante : $u_0 = 0$, $u_1 = 10$ car lorsqu'on lit la valeur de u_0, on dit qu'il y a 1"0", $u_2 = 1110$ car u_1 est constitué de 1"1" et 1"0, $u_3 = 3110$ etc.

1. Quelle instruction transforme un entier n en la chaîne de caractères de l'écriture décimale de n ?
2. Écrire une fonction `Robinson`(n) d'argument un entier n qui renvoie la valeur de u_n.
3. Écrire une fonction `long_Rob`(n) d'argument un entier n qui renvoie le nombre de chiffres de l'écriture décimale de u_n.
4. Représenter sous Python le nombre de chiffres de l'écriture décimale de u_n en fonction de n.

Exercice 48

Un administrateur d'un site web veut assurer un maximum de sécurité pour les utilisateurs du site.

Pour ceci il décide de réaliser une application qui évalue la force des mots de passe des différents utilisateurs du site, sachant qu'un mot de passe est une chaine de caractères qui ne comporte pas d'espaces et de lettres accentuées.

La force d'un mot de passe varie, selon la valeur d'un score calculé, de 'Très faible' jusqu'à 'Très fort' selon le barème suivant :

score <20 : Très faible, 20 ⩽ score < 40 : Faible, 40 ⩽ score < 80 : Fort, Sinon :Très fort.

Le score se calcule en additionnant des bonus et en retranchant des pénalités définis par :

Les bonus attribués sont :

4*Nombre total de caractères

+ 2*(Nombre total de caractères – nombre de lettres majuscules)

+ 3*(Nombre total de caractères – nombre de lettres minuscules)

+ 5*Nombre de caractères non alphabétiques.

Les pénalités imposées sont :

2*longueur de la plus longue séquence de lettres minuscules

+ 3* longueur de la plus longue séquence de lettres majuscules

Exemple pour le mot de passe 'P@cSI_promo2017' :

Le nombre total de caractères est 15, le nombre de lettres majuscules est 3, le nombre de lettres minuscules est 6, le nombre de caractères non alphabétiques est 6 donc les bonus valent 4*15+2*(15-3)+3*(15-6)+6*5 = 141.

La longueur de la plus longue séquence de lettres minuscules('promo') est 5 et la longueur de la plus longue séquence de lettres majuscules('SI') est 2 donc les pénalités valent 2*5+2*2 = 14.

Le score final est 141 - 14 = 127 > 80 donc le mot de passe est 'Très fort'.

1. Écrire une fonction `Nbmin`(*pass*) qui retourne le nombre de caractères minuscules de la chaîne de caractères *pass* (un élément *el* est une minuscule si $'a' \leqslant el \leqslant 'z'$).
2. Écrire une fonction `NbMaj`(*pass*) qui retourne le nombre de caractères majuscules de la chaîne *pass*.
3. Écrire une fonction `NbnonAlpha`(*pass*) qui retourne le nombre de caractères non alphabétiques de la chaîne *pass*.
4. Écrire une fonction `LongMaj`(*pass*) qui retourne la longueur de la plus longue séquence de lettres majuscules de la chaîne *pass*.
5. Écrire une fonction `Longmin`(*pass*) qui retourne la longueur de la plus longue séquence de lettres minuscules de la chaîne *pass*.
6. Écrire une fonction `Score`(*pass*) qui affiche le score du mot de passe correspondant à la chaîne *pass*.

Exercice 49

Le jeu de Nim se joue à deux joueurs. On dispose initialement d'une rangée de *n* allumettes ; chacun à son tour, les joueurs doivent retirer entre 1 et 3 allumettes. Le joueur qui a retiré la dernière allumette a perdu.

1. Écrire une fonction `entree` qui permet de demander à l'utilisateur de saisir au clavier un nombre entre 1et 3, qui repose la même question tant que l'entier entré ne convient pas et renvoie cet entier.
2. Simulation d'un tour entre l'utilisateur et l'ordinateur : l'utilisateur retire le nombre qu'il a saisi au clavier puis l'ordinateur enlève le complément à 4 de ce que l'utilisateur a retiré (1 pour 3, 2 pour 2, 3 pour 1) sauf s'il ne reste plus que 3 allumettes dans ce cas l'ordinateur en retire 2 ou s'il en reste 2, il en retire 1.

   Écrire une fonction `tour` d'argument une rangée d'allumettes permettant de modéliser un tour de jeu (le coup de l'utilisateur et celui de l'ordinateur).

   S'il ne reste plus d'allumettes après le coup de l'utilisateur, la fonction affichera "*PERDU*" , si c'est après le coup de l'ordinateur, elle affichera "*GAGNE*" .
3. Écrire une fonction `Nim`(*n*) prenant en argument un entier *n* égal au nombre initial d'allumettes et permettant de jouer une partie de ce jeu contre l'ordinateur.