Tous les exercices sont précédés de la consigne ci-après : *Cet exercice devra être fait avec le langage Python. À chaque question, les instructions ou les fonctions écrites devront être testées.*

Exercice 1 On considère une pièce de monnaie équilibrée, c'est-à-dire donnant à chaque lancer Pile (1) ou Face (0) avec la même probabilité $p = \frac{1}{2}$ (Loi de Bernoulli).

1. Écrire une fonction `tirer` d'argument un entier n et renvoyant la série de résultats (1 ou 0) obtenus en simulant la succession de n lancers de la pièce. On pourra utiliser la fonction `randint` du module `random`.

2. Soit une liste de zéros et de uns obtenue par n lancers indépendants numérotés de 0 à $(n-1)$. On s'intéresse aux "séries", c'est-à-dire aux successions de lancers pour lesquels la pièce tombe du même côté. Par exemple la suite de lancers 11101001111 contient successivement les séries : 111, 0, 1, 00, 1111. La longueur d'une série est le nombre de lancers qu'elle contient. Dans notre exemple, les longueurs sont : 3,1,1,2,4.
   Créer une fonction `longueursSeries` dont l'argument est une liste `L` de zéros et de uns et qui renvoie la suite des longueurs des séries obtenues, dans l'ordre. Par exemple, `L =[1,1,1,0,1,0,0,1,1,1,1]` donnera `[3,1,1,2,4]`. Noter que la somme des longueurs des séries vaut n, la longueur de `L`.

3. En déduire une fonction `tirerSeries` d'argument un entier n et renvoyant la liste des longueurs des séries issues d'une simulation de n lancers successifs.

4. On souhaite maintenant compter le nombre de séries au fur et à mesure qu'elles apparaissent lors des lancers. Par exemple :

Numéro i du lancer :	0	1	2	3	4	5	6	7	8	9	10
Valeur du lancer :	1	1	1	0	1	0	0	1	1	1	1
Nombre N_i de séries obtenu :	1	1	1	2	3	4	4	5	5	5	5

   Écrire une fonction `compter` d'argument une liste s de longueurs de séries et qui renvoie la liste $[N_1, N_2, \cdots, N_n]$, où N_i est le nombre de séries obtenu après i lancers. Par exemple, `[3,1,1,2,4]` donnera `[1,1,1,2,3,4,4,5,5,5,5]`.

5. À partir des fonctions construites, mettre en oeuvre une démarche permettant d'émettre une conjecture sur le nombre moyen de séries lors d'une succession de n lancers.

Exercice 2 Pour tout entier naturel n non nul, on note E_n la liste $[1, \cdots, n]$ des entiers de 1 à n. On représente une partie A de E_n par la liste $[\alpha_1, \cdots, \alpha_p]$, où $\alpha_1 < \cdots < \alpha_p$. Par convention, la partie vide est représentée par la liste vide [].

1. On considère la fonction suivante :

```
1 def A(listes,numero):
2     resultat = []
3     for L in listes :
4         resultat.append( L + [numero] )
5     return resultat
```

   Quel est le type de l'argument `listes` ? Que renvoie la fonction `A` ?

On représente un ensemble de parties de E_n par une liste de listes représentant ces parties. Ainsi l'ensemble $\{\emptyset, \{2\}, \{3,4\}\}$ est représenté, par exemple, par $[[], [2], [3,4]]$.

2. Donner la liste `L0` représentant l'ensemble des parties de E_5 à 0 élément.

3. Écrire une fonction récursive `parties` de deux arguments n et p renvoyant la liste représentant l'ensemble des parties de E_n ayant au plus p éléments.
On pourra remarquer que l'ensemble des parties à au plus p éléments de E_n se partitionne en l'ensemble des parties à au plus p éléments de E_n contenant n et l'ensemble des parties à au plus p éléments de E_n ne contenant pas n.

4. Donner la liste `L1` représentant l'ensemble des parties de E_5 à 1 élément, et celle `L5` représentant l'ensemble des parties de E_5 à 5 éléments.

5. Écrire une fonction récursive `parties2` de deux arguments n et p renvoyant la liste représentant l'ensemble des parties de E_n à p éléments.

Exercice 3 Pour un entier naturel non nul n, son nombre de diviseurs est le nombre d'entiers naturels inférieurs ou égaux à n qui le divisent. Si cet entier n a strictement plus de diviseurs que chacun des entiers naturels qui le précèdent, on dit que c'est un nombre riche. C'est un peu le contraire d'un nombre premier qui n'a que deux diviseurs.
L'objet de cet exercice est de créer la liste des premiers nombres riches puis d'examiner leurs facteurs premiers.

1. Écrire une fonction `nbdiv` d'argument un entier naturel n qui renvoie le nombre de diviseurs de n. La tester pour n = 60.

2. Écrire une fonction `riches` d'argument un entier naturel K qui renvoie les K premiers nombres riches sous forme d'une liste de couples (n, d), où d est le nombre de diviseurs de n. Par exemple, `riches(3)` doit donner `[[1,1],[2,2],[4,3]]`. Faire afficher les 16 premiers nombres riches.

3. Écrire une fonction `facteursPremiers` d'argument un entier naturel n qui renvoie la liste des facteurs premiers de n, avec répétitions si nécessaire. Par exemple, `facteursPremiers(360)` doit renvoyer `[2,2,2,3,3,5]`.

4. Modifier la fonction `facteursPremiers` de sorte qu'elle renvoie la décomposition en facteurs premiers de n sous la forme d'une liste de couples (p, i), où i est le nombre de fois où apparaît le facteur premier p dans la décomposition. Par exemple, `facteursPremiers(360)` doit maintenant renvoyer `[[2,3],[3,2],[5,1]]`.

5. En utilisant la fonction `facteursPremiers` sur les premiers nombres riches, que peut-on conjecturer sur le lien entre la décomposition en facteurs premiers et le nombre de diviseurs ?

Exercice 4 Pour tout ensemble non vide, identifié ici à une liste d'éléments deux à deux distincts, $E = [e_0, \ldots, e_{n?1}]$ de taille n, chacune de ses parties A peut être codée sous forme d'une liste `C` à n éléments contenant des zéros et des uns :

`C[i] = 1` si $e_i \in A$, et `C[i] = 0` sinon.

Par exemple, les parties $\emptyset$, $[a]$, $[b]$ et $[a, b]$ de la liste $[a, b]$ sont respectivement codées par les listes $[0, 0]$, $[1, 0]$, $[0, 1]$ et $[1, 1]$.

1. Écrire une fonction `decoder` de deux arguments E et C renvoyant la partie de la liste E codée par le code C.
Ainsi, `decoder([2,3,5,7],[1,0,0,1])` donne `[2,7]` ; de même `decoder([2,3,5,7],[0,0,0,0])` donne `[]`.

2. Écrire une fonction `coder` de deux arguments E et A renvoyant le code de la partie A de la liste E. Ainsi `coder([2,3,5,7],[2,7])` donne `[1,0,0,1]`.

3. Écrire une fonction `incrementer` d'argument une liste C de zéros et de uns de taille n, représentant sur n bits l'écriture en base 2 d'un entier naturel k, et renvoyant la liste représentant sur n bits l'écriture en base 2 de l'entier $(k+1)$.
Par exemple, `[0,1,0,1,1,1]` est l'écriture en base 2 sur 6 bits de 23 et `[0,1,1,0,0,0]`, l'écriture en base 2 sur 6 bits de 24.

4. En déduire la fonction `parties` d'argument E renvoyant la liste des parties de la liste E.

5. Écrire une fonction `p_parties` de deux arguments, un ensemble E non vide de taille n et un entier naturel $p \leqslant n$, qui renvoie la liste des parties de E de taille p.

Exercice 5 L'objet de cet exercice est d'étudier $q(n)$ le nombre de triplets (x, y, z) d'entiers naturels tels que :

$$x + 2y + 3z = n.$$

1. Programmer une fonction `p` d'argument m renvoyant le nombre de couples (x, y) d'entiers naturels tels que $x + 2y = m$. On distinguera les cas "m pair" et "m impair".

2. En déduire une fonction `q1` d'argument n renvoyant $q(n)$. Calculer $q(n)$ pour n entier de 0 à 10, ainsi que $q(100)$.

3. Au brouillon, exprimer $q(n+3)$ en fonction de $q(n)$ et de $p(n+3)$.
En déduire que $q(n+6) = q(n) + n + 6$.

4. En déduire une fonction récursive `q2` d'argument n renvoyant $q(n)$.

5. On peut démontrer que $q(j+6k) = q(j) + \sum_{i=1}^{k}(j+6i) = q(j) + jk + 3k(k+1)$.
En déduire une troisième fonction `q3` d'argument n renvoyant $q(n)$.

6. Quels sont les coûts de calcul des fonctions `q1`, `q2`, `q3` ? Conclure.

Exercice 6 Soit une liste L de longueur paire $2n$. L est dite de type $\mathcal{D}$ si et seulement si :

- L ne contient que des 1 et des -1 ;
- L contient autant de 1 que de -1 ;
- chaque sous-liste d'éléments consécutifs de L en partant du début contient au moins autant de 1 que de -1.

Par convention, la liste vide est de type $\mathcal{D}$.

1. Quel est nécessairement le premier élément d'une liste de type $\mathcal{D}$? Au brouillon, donner toutes les listes de type $\mathcal{D}$ de longueurs 2 et 4.

2. Écrire une fonction `D2par` d'argument une liste L qui ne contient que des 1 et des -1, qui renvoie une chaîne de caractères obtenue en convertissant chaque 1 de L en parenthèse ouvrante `"("` et chaque -1 en parenthèse fermante `")"`, et en mettant bout-à-bout les caractères ainsi obtenus.
Tester cette fonction sur les listes de type $\mathcal{D}$ de longueurs 2 et 4.

3. Écrire une fonction `DQ` d'argument une liste L qui renvoie un booléen indiquant si L est de type $\mathcal{D}$ ou pas.

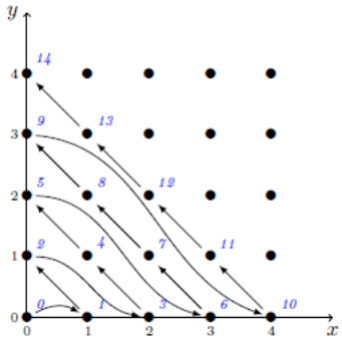
4. On admet que toute liste de type $\mathcal{D}$ non vide s'écrit de façon unique sous la forme `[1]+u+[-1]+v`, où `u` et `v` sont deux listes de type $\mathcal{D}$, éventuellement vides.
   Écrire une procédure récursive `LD` d'argument un entier naturel N qui renvoie la liste de toutes les listes de type $\mathcal{D}$ de longueur $2N$.

Exercice 7 Soit une matrice carrée M d'ordre 9 dont chaque coefficient est repéré par un numéro de ligne i et un numéro de colonne j, compris entre 0 et 8. On attribue à chacun de ces coefficients un seul numéro $n(i,j)$ défini par le code *Python* suivant :

```
def n(i,j):
    if j == 0 :
        if i == 0 :
            return 0
        else :
            return n(i-1,8)+1
    else :
        return n(i,j-1)+1
```

1. Expliquer concrètement comment la fonction `n` numérote les coefficients de M.
2. Déterminer l'expression de $n(i,j)$ en fonction de i et de j.
3. Écrire une fonction `ij` d'argument m qui renvoie le couple (i,j) tel que $n(i,j) = m$.

On se place maintenant dans un plan muni d'un repère orthonormé, et on numérote chaque point de coordonnées (x,y) où x et y sont des entiers naturels par le procédé décrit sur la figure ci-dessous :



4. Écrire une fonction récursive `num1` d'arguments x et y ayant pour résultat le numéro du point de coordonnées (x,y).
5. Écrire une fonction non récursive `num2` faisant la même chose.
6. Déterminer la fonction réciproque de `num1` (ou de `num2`).

Exercice 8 On considère un jeu de 32 cartes. Il est formé de couples de 8 valeurs ordonnées, `valeurs=["7", "8", "9", "10", "V", "D", "R", "A"]`, et de 4 "couleurs", `couleurs=["T", "K", "C", "P"]` (Trèfle, Carreau, Coeur, Pique). On distribue au hasard une "main", c'est-à-dire 5 cartes distinctes, et on s'intéresse à des mains particulières :

- les "une paire" (2 cartes de même valeur et 3 cartes de valeurs différentes) ;
- les "deux paires" (2 cartes d'une même valeur, 2 cartes d'une même autre valeur, et une carte d'une troisième valeur) ;

- les "fulls" (3 cartes d'une même valeur et 2 cartes d'une même autre valeur) ;
- les "carrés" (les 4 cartes d'une même valeur, et une autre).

1. Construire la liste cartes des 32 cartes à partir des deux listes `valeurs` et `couleurs`, chaque carte étant représentée par la paire $[valeur, couleur]$. Vérifier que le nombre de cartes obtenu est correct.

2. Écrire une fonction `tirerMain` sans argument qui renvoie une liste de 5 cartes distinctes tirées au hasard. On pourra utiliser la fonction `sample` du module `random`.

3. Écrire une fonction `LV` d'argument une main et qui renvoie la liste croissante des nombres de valeurs identiques dans la main. Par exemple, si la main est "une paire", la fonction `LV` renvoie `[1,1,1,2]`. On pourra utiliser la fonction intrinsèque `sorted`.

4. À partir de 50000 tirages aléatoires de mains, estimer les probabilités d'obtenir une "une paire", "deux paires", un full et un carré.

5. Comparer ces estimations avec les probabilités obtenues par dénombrement. On pourra utiliser la fonction `comb` du module `scipy.special`, ou faire autrement.

Exercice 9

1. Écrire une fonction `sp` d'argument une liste L qui renvoie la liste obtenue en prenant d'abord le dernier terme de L, puis le premier, puis l'avant-dernier, puis le deuxième, puis l'antépénultième, puis le troisième, *etc.* (On parle de permutation en *spirale*). Vérifier que `sp([1,2,3,4,5,6])` donne bien `[6,1,5,2,4,3]`, et que `sp([1,2,3,4,5,6,7])` donne `[7,1,6,2,5,3,4]`.

2. Vérifier que si l'on itère la fonction `sp` sur $L = [1, 2, 3, 4, 5, 6]$ on retombe sur L à la sixième itération. Qu'en est-il pour $L = [1, 2, 3, 4, 5, 6, 7]$?

3. Écrire une fonction `periode` d'argument un entier n qui renvoie le nombre minimum p d'itérations de la fonction `sp` pour retomber sur la liste $[1, 2, \cdots, n]$, en partant de cette même liste.

4. Déterminer les entiers inférieurs ou égaux à 99 tels que `periode`$(n) = n$.

5. À l'aide des instructions `plot` et `show` du module `matplotlib.pyplot`, faire tracer `periode`$(n)/n$ en fonction de n, pour n compris entre 1 et 99. Déterminer la valeur de n qui minimise `periode`$(n)/n$.

Exercice 10 Soit p un nombre premier. On note E_p l'ensemble des entiers naturels compris entre 1 et $(p-1)$. Pour deux éléments x et y de E_p, on appelle alors "*produit de x et y dans E_p*", noté $[xy]_p$, le reste de la division euclidienne de xy par p, ce reste étant nécessairement non nul. Le "*carré de x dans E_p*", noté $[x^2]_p$, sera ainsi défini comme le produit de x par x dans E_p. Plus généralement, pour un entier naturel k non nul, $[x^k]_p$ sera défini par récurrence : $[x^1]_p = x$ et, pour tout k dans $\mathbb{N}^*$, $[x^{k+1}]_p$ est le produit de x par $[x^k]_p$ dans E_p.

1. Lorsque $p = 17$, construire de proche en proche la liste des p premiers $[2^k]_p$ et l'afficher.

2. Pour chaque $x \in E_p$, l'ensemble des $[x^k]_p$ pour $k \in \mathbb{N}$ est fini car il est inclus dans E_p.
   Écrire une fonction `orb` de deux arguments p et x qui renvoie la liste L des $[x^k]_p$ pour k entier de 1 à n, où n est l'unique valeur telle que la liste L ne contient que des valeurs distinctes et $[x^{n+1}]_p \in L$. Faire afficher `orb(17,x)` pour tous les éléments x de E_{17}. Que constatez-vous ?

3. On admettra que quel que soit le nombre premier p, il existe un élément g de E_p tel que l'ensemble des $[g^k]_p$ pour $1 \leqslant k \leqslant p$ soit exactement l'ensemble E_p. On dit que g *est un générateur de* E_p.
   En utilisant la propriété admise que g est un générateur de E_p si et seulement si $p-1$ est le plus petit entier non nul k tel quel $[g^k]_p = 1$, écrire une fonction `ppg` d'argument un nombre premier p qui renvoie le *plus petit générateur* de E_p.
   Déterminer le plus petit générateur de E_p pour $p = 17$, puis pour $p = 106031$.

4. Si g est un générateur de l'ensemble E_p, indiquer pourquoi l'application $x \mapsto [g^x]_p$ définit une bijection de E_p dans lui-même.
   Écrire ensuite une fonction `recip` de trois arguments, un nombre premier p, un générateur g de E_p et y un élément de E_p, qui renvoie l'élément x de E_p tel que $y = [g^x]_p$. La recherche de y se fera par essais successifs des éléments de E_p.
   Dans le cas $p = 106031$, déterminer l'antécédent de $y = 2$.

5. Quel est le coût de la fonction `recip` dans le meilleur des cas, dans le pire des cas ?

Exercice 11

1. Le fichier `algo118-pi-digits.txt` se trouvant dans le répertoire `data` contient les premières décimales de π, sur une seule ligne et sans espace entre les chiffres.
   Récupérer le contenu de ce fichier sous forme d'une chaîne de caractères que l'on nommera `decpi`.

2. Faire afficher les 10 premiers caractères de `decpi`, ses `10` derniers, ainsi que le nombre `nbdec` de caractères de `decpi`.

3. Écrire une fonction `trouve` de deux arguments, deux chaînes de caractères P et M, qui renvoie un entier naturel p si M est une sous-chaîne de P commençant à la position p, et -1 si M n'est pas une sous-chaîne de P. On n'utilisera pas la méthode `find` de la classe `str`.
   Par exemple, `trouve("BanquePT","an")` donne `1`, `trouve("BanquePT","PT")` donne `6`, alors que `trouve("BanquePT","PSI")` donne `-1`. Comparer avec `"BanquePT".find("an")`, *etc.*

4. Les nombres `"314159"`, `"123456"`, `"12345"`, et `"1789"` se trouvent-ils dans `decpi`, et si oui, en combien d'exemplaires et à quelle(s) position(s) ?

5. Le fichier `algo118-pi-false.txt` se trouvant dans le répertoire `data` contient les premières décimales de π, avec quelques chiffres erronés. Combien et en quelles positions ?

Exercice 12 On appelle *entier palindrome* un entier naturel non nul qui est égal à l'entier obtenu en renversant l'ordre de ses chiffres. Par exemple, 22, 121 et 2552 sont des entiers palindromes et 13, 211 et 2525 n'en sont pas.
Attention, il ne peut y avoir de 0 comme premier ou dernier chiffre d'un entier palindrome.

1. Après avoir observé dans la console ce que donne `list(str(123))`, écrire une fonction `isp` d'argument un entier naturel non nul qui renvoie un booléen indiquant si l'entier est un palindrome, ou pas.
   Par exemple, `isp(1245421)` donne `True` alors que `isp(1245422)` donne `False`.

2. Afficher la liste de tous les palindromes compris entre 1 et 1000, obtenue grâce à la fonction `isp`, ainsi que le nombre de ses éléments.

3. Écrire une fonction `palp` d'argument un entier naturel non nul p qui renvoie la liste de tous les palindromes à $2p$ chiffres, construite à partir des nombres de p chiffres.
   Vérifier que 1001, 2002, 3003 sont éléments de la liste `palp(2)`, qui doit comporter 90 éléments.

4. En déduire une fonction `palindromes` d'argument un entier naturel non nul n qui renvoie la liste de tous les palindromes à n chiffres.

5. Afficher tous les entiers palindromes d'au plus 8 chiffres, de carrés palindromes.
   Que peut-on conjecturer ?

6. Écrire une fonction `postulants` d'arguments un entier naturel non nul p, qui renvoie la liste des palindromes d'au plus $2p$ chiffres qui ne contiennent que des 0 et des 1, avec éventuellement un 2 en position centrale ou aux extrémités.
   Afficher le nombre d'éléments de la liste `postulants(6)`, ainsi que le nombre d'éléments de celle-ci dont les carrés ne sont pas des palindromes.

Exercice 13 On note $\lfloor u \rfloor$ la partie entière de u. Soit x un réel. On lui associe les suites $(a_n)_{n\geqslant 0}$ et $(v_n)_{n\geqslant 0}$ définies par

$$v_0 = x, \quad a_n = \lfloor v_n \rfloor, \quad \text{et} \quad v_{n+1} = \begin{cases} \dfrac{1}{v_n - a_n} & \text{si } v_n \neq a_n \\ 0 & \text{sinon} \end{cases} \quad \text{pour } n \geqslant 0.$$

1. Calculer les 10 premiers termes de la suites $(a_n)_{n\geqslant 0}$ pour $x = \sqrt{3}$.

2. Écrire une fonction `A` qui, à un réel x et un entier N, associe les N premiers termes de la suite. On testera les valeurs $x = 2/3$, $\frac{\sqrt{5}-1}{2}$, e, e^2, π.

3. Écrire une fonction `ND` de trois arguments entiers a, n et d qui renvoie le numérateur et le dénominateur de $a + \dfrac{d}{n}$ (sans simplification éventuelle).

4. En déduire une fonction `frac` qui, à un réel x et un entier N, associe le numérateur et le dénominateur de la fraction

$$c_N = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots \; a_{r-2} + \cfrac{1}{a_{r-1} + \cfrac{1}{a_r}}}}}$$

   où les a_i dépendent de x et où r est le plus grand entier inférieur ou égal à N tel que $a_r \neq 0$.

5. On admet que la suite $(c_n)_{n\geqslant 0}$ tend vers x (elle peut être constante à partir d'un certain rang). Écrire un programme qui à un réel x et un réel positif ε associe un couple (p, q) tel que

$$\left| x - \frac{p}{q} \right| < \varepsilon.$$