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 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 perpendiculaire. Si le cavalier est loin des bords de l'échiquier, il y a 8 possibilités de déplacements, mais il en a moins s'il est près des bords.

1. Illustrer sur un brouillon les deux cas énoncés précédemment.
2. Écrire une fonction `valide` prenant en argument deux entiers relatifs i et j et vérifiant que le couple $[i, j]$ est bien une position de l'échiquier. `valide` doit renvoyer un booléen.
3. Écrire une fonction `coupsSuivants` prenant en argument une position $[i, j]$ et renvoyant la liste des positions que peut atteindre un cavalier placé en $[i, j]$ en un seul coup.
4. Écrire une fonction `cavalier` prenant en argument une position $[a, b]$ et renvoyant 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 $[i, j]$.

Exercice 2

1. On considère un nombre n. Que donne la commande `list(str(n))` ?
2. Comment récupérer le chiffre des unités d'un nombre entier donné ? Celui des dizaines ? Tester cette méthode sur le nombre 2015.
3. Écrire une fonction `transforme` d'argument un entier naturel n et qui renvoie le nombre obtenu par la méthode suivante :
   - les chiffres de rang impair (en partant des unités) sont inchangés.
   - les chiffres de rang pair (en partant des unités) sont doublés, si ce double est supérieur ou égal à 10, on le remplace par la somme de ses chiffres (par exemple, 3 est remplacé par 6 et 7 est remplacé par 5).

   Exemple : 43281 est transformé en 46271.
4. Déterminer les nombres inférieurs à 10000 invariants par la fonction `transforme`. Expliquer le résultat obtenu.

Exercice 3 Dans cet exercice, un segment $[a, b]$ $(a \leqslant b)$ est représenté par une liste de taille 2 : `[a,b]`.

1. Deux segments sont disjoints si leur intersection est vide. Écrire une fonction `disjoints` de deux arguments `i1` et `i2` qui teste si les segments `i1` et `i2` sont disjoints. La fonction renvoie le booléen `True` s'ils sont disjoints, `False` sinon.
2. La fusion de deux segments a comme minimum le plus petit des minima des deux segments, et comme maximum le plus grand des maxima des deux segments. Écrire une fonction `fusion` de deux arguments `i1` et `i2` et qui renvoie le segment correspondant à la fusion de `i1` et `i2`.

Une "*liste bien formée*" est une liste de segments qui vérifie les propriétés suivantes :

- les segments sont deux à deux disjoints ;

- les segments de la liste sont classés par ordre croissant, en considérant qu'un segment `i1` est strictement plus petit qu'un segment `i2` si et seulement si le maximum de `i1` est strictement inférieur au minimum de `i2`.

3. a) Les listes suivantes sont-elles bien formées ?
   - `L1=[[0,1],[2,5],[3,6]]`
   - `L2=[[2,5],[0,1],[3,6]]`
   - `L3=[[0,1],[2,3],[4,6]]`

   b) Écrire une fonction récursive `verifie` qui teste si une liste de segments est bien formée.

4. Écrire une fonction récursive `appartient` de deux arguments `x` et `L` qui teste si la valeur `x` est élément des segments de la liste `L` bien formée.

Exercice 4 Dans cet exercice, un segment $[a, b]$ ($a \leqslant b$) est représenté par une liste de taille 2 : `[a,b]`.

1. Deux segments sont disjoints si leur intersection est vide. Écrire une fonction `disjoints` de deux arguments `i1` et `i2` qui teste si les segments `i1` et `i2` sont disjoints. La fonction renvoie le booléen `True` s'ils sont disjoints, `False` sinon.

2. Une "*liste bien formée*" est une liste de segments qui vérifie les propriétés suivantes :

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

   Écrire une fonction `decouper` de deux arguments, une valeur `x` et une liste de segments `L` bien formée, qui renvoie un triplet décomposé de :

   - une liste bien formée contenant les segments strictement inférieurs à la valeur `x` ;
   - une liste bien formée contenant, s'il existe, le segment contenant la valeur `x`, et sinon, une liste vide ;
   - une liste bien formée contenant les segments strictement supérieurs à la valeur `x`.

3. Écrire une fonction `inserer` de deux arguments, un segment `i` et une liste de segments `L` bien formée. Cette fonction renvoie une liste bien formée de segments contenant les segments de `L` qui sont disjoints de `i` et :

   - soit l'intervalle `i` si tous les segments contenus dans `L` lui sont disjoints ;
   - soit le segment union de `i` et des segments contenus dans `L` qui ne sont pas disjoints de `i`.

Exercice 5 Dans cet exercice, on manipule des suites (finies) d'entiers sous la forme de listes d'entiers. Ainsi la suite $(0, 1, 2, 3)$ sera représentée par la liste `[0,1,2,3]`.
La liste est croissante (respectivement décroissante, monotone) si la suite est croissante (respectivement décroissante, monotone).

1. Écrire une fonction `estCroissante` qui teste si une liste d'entiers est croissante. Cette fonction devra être de complexité linéaire.

2. Écrire de même une fonction `estDecroissante` qui teste si une liste d'entiers est décroissante.

3. Écrire de même une fonction `estMonotone` qui teste si une liste d'entiers est monotone.

4. Soit la liste $L = [u_0, u_1, \cdots, u_{n-1}]$ de longueur n. Une monotonie de L est un couple d'indices (i, j) tel que $0 \leqslant i < j < n$, que la sous-liste $[u_i, u_{i+1}, \cdots, u_j]$ est monotone et qu'elle ne l'est plus si on l'étend, à droite ou à gauche, d'un élément supplémentaire (lorsque c'est possible). La monotonie est dite "banale" lorsque $j = i + 1$.

   a) Proposez une liste d'entiers de longueur 5 qui ne présente que des monotonies banales. Peut-on avoir deux termes consécutifs égaux dans une liste ne présentant que des monotonies banales ?

   b) Écrire une fonction `cahots`, de complexité linéaire, qui teste si une liste ne comporte que des monotonies banales.

   c) Après avoir créé une liste arbitraire `L` de valeurs toutes distinctes, on peut l'ordonner par `L.sort()`. Imaginer ensuite une méthode pour réordonner de manière à ce qu'elle ne comporte que des monotonies banales.

Exercice 6

1. **Compréhension de code :** que fait le code *Python* suivant ? Quelle est l'une de ses particularités ?

```
def ordonner(L) :
    if len(L) <= 1 :
        return L
    e0 = L[0]
    n = 1
    Linf, Lsup = [], []
    for ei in L[1:]:
        if ei == e0 :
            n += 1
        elif ei < e0 :
            Linf.append(ei)
        else :
            Lsup.append(ei)
    return ordonner(Linf)+n*[e0]+ordonner(Lsup)
```

2. Écrire une fonction `less` de deux nombres complexes `z1` et `z2` qui renvoie `True` si et seulement si $\mathcal{R}e(z_1) < \mathcal{R}e(z_2)$ ou "$\mathcal{R}e(z_1) = \mathcal{R}e(z_2)$ et $\mathcal{I}m(z_1) < \mathcal{I}m(z_2)$".

3. En s'inspirant du code définissant la fonction `ordonner`, écrire puis tester une fonction `ordonnerDansC` qui ordonne une liste de complexes selon la relation d'ordre définie précédemment.

4. Tester la fonction `ordonnerDansC` sur une liste de complexes dont les parties réelles et imaginaires sont des entiers tirés au hasard entre -10 et 10. On pourra utiliser l'instruction `randint` du module `random`.

5. En utilisant le module `cmath`, créer la liste des $(20 + \cos(10a))\exp(ia)$, pour a variant de $-\pi$ à π avec un pas de $\pi/100$. Ordonner ces points par `ordonnerDansC`. Faire tracer dans le plan complexe le nuage de points ainsi que la ligne les reliant. On utilisera `plot`, `axis` et `show` du module `matplotlib.pyplot` et le repère sera rendu orthonormé par `axis('equal')`.

Exercice 7 Soit n un entier supérieur à 2. On considère l'ensemble S_n des bijections de l'ensemble $\{0, 1, 2, \cdots, n-1\}$ dans lui-même.
On code une bijection f par la liste des n premiers entiers donnée par $[f(0), f(1), \cdots, f(n-1)]$. Par exemple, la liste $[3, 0, 4, 1, 2, 5]$ code $\phi \in S_6$ la bijection $\phi : \{0, 1, 2, 3, 4, 5\} \to \{0, 1, 2, 3, 4, 5\}$ définie par

$$\phi(0) = 3, \quad \phi(1) = 0, \quad \phi(2) = 4, \quad \phi(3) = 1, \quad \phi(4) = 2 \quad \text{et} \quad \phi(5) = 5.$$

1. Soit f une bijection codée par une liste `L`. Que vaut $f(i)$?
2. Écrire une fonction `rond` qui reçoit en arguments deux listes `L1`, `L2` (codant deux bijections f et g de S_n) et renvoie la liste codant $f \circ g$.
   On définit l'application $\varepsilon : S_n \to \mathbb{Q}$ en posant
   $$\varepsilon(f) = \prod_{0 \leqslant i < j \leqslant n-1} \frac{f(i) - f(j)}{i - j}.$$
3. Coder ε.
4. Que fait l'instruction `sample` du module `random` ?
5. Vérifier pour quelques valeurs de n et sur 10 bijections f de S_n choisies aléatoirement que l'on a $\varepsilon(f) = \pm 1$.
6. En s'inspirant de la méthode de la question précédente, conjecturer une relation entre $\varepsilon(f \circ g)$, $\varepsilon(f)$ et $\varepsilon(g)$.

Exercice 8 On souhaite créer la liste de tous les nombres premiers inférieurs ou égaux à un certain nombre n, puis tester la primalité d'un entier naturel.

1. Créer une liste `L30` de 31 booléens `True` (indices de 0 à 30), qui servira de liste-test.

   On va créer une liste L de booléens contenant des `False` aux indices qui ne sont pas premiers et des `True` aux indices qui le sont. Il faudra donc modifier tous les éléments de la liste L dont l'indice n'est pas premier.
2. Écrire une fonction `modifier` de deux arguments, une liste L de booléens et un entier naturel p, qui modifie la liste L de la façon suivante : Si p vaut zéro, met le premier élément de la liste L à `False` ; si p vaut un, met le deuxième élément de la liste à `False` ; sinon, met à `False` tous les éléments de L dont l'indice est un multiple de p autre que p. Noter que cette fonction ne renvoie rien.
3. En déduire une fonction `premiers` d'argument n renvoyant la liste des nombres premiers inférieurs ou égaux à n. Rappelons qu'un nombre inférieur ou égal à n est premier s'il n'est pas divisible par aucun entier inférieur ou égal à $\sqrt{n}$.
4. Tester la primalité des nombres suivants en minimisant le temps de calcul :
   `696937, 4862592, 9412213, 39841247, 99770809, 5263996587, 2358869562457`.

Exercice 9 Soit la suite $(u_n)_{n \in \mathbb{N}}$ de nombres entiers dépendant de sa valeur initiale $u_0 \in \mathbb{N}^*$ et définie par :
$$\forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} u_n/4 \text{ si } u_n \text{ est un multiple de 4} \\ (3u_n + 2)/2 \text{ sinon (division entière)} \end{cases}$$

1. À l'aide notamment des instructions `plot` et `show` de la bibliothèque `mathplotlib.pyplot`, représenter graphiquement les 30 premiers termes de la suite pour $u_0 = 25$, puis pour $u_0 = 37$.
2. On conjecture que pour tout entier naturel u_0 non nul, il existe un plus petit entier n, appelé "durée de vol", tel que $u_n = 1$.
   Que se passe-t-il s'il existe $n \in \mathbb{N}$ tel que $u_n = 1$?
3. Écrire une fonction `vol` d'argument un entier d, renvoyant la durée de vol lorsque $u_0 = d$ ainsi que la valeur maximale atteinte par la suite, que l'on appellera "altitude".

4. Représenter cette altitude en fonction du point de départ d, pour les valeurs de d inférieures à 1000.

5. Pour quelles valeurs de u_0 inférieures à 1000, atteint-on une altitude supérieure à 100 000 ? Donner alors les valeurs de u_0, de la durée de vol et de l'altitude.

6. Vérifier que ces valeurs correspondent aussi à des durées de vol maximales.

Exercice 10 On considère les triangles de nombres de type

```
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*.

1. Écrire une fonction `suivante` d'argument une liste de zéros et de uns correspondant à une ligne et renvoyant la liste représentant la ligne suivante.

2. Écrire une fonction `triangle` d'argument une liste donnant la première ligne et renvoyant la liste de listes représentant le triangle. La tester avec la première ligne de l'exemple.

3. Écrire une fonction `afficher` d'argument une liste de listes représentant un triangle, qui ne renvoie rien mais qui affiche dans la console le triangle correspondant, comme dans l'exemple ci-dessus.

4. Écrire une fonction `equilibreQ` d'argument un triangle et testant si ce triangle est équilibré.

5. Déterminer le nombre de triangles équilibrés ayant une première ligne de 4 nombres.

6. Écrire une fonction `nbteq` d'argument n renvoyant le nombre de triangles équilibrés ayant une première ligne de n nombres.
   Faire afficher le résultat pour n variant de 1 à 15.

Exercice 11 Pour un entier naturel n non nul fixé, on appelle *vecteur creux* de $\mathbb{R}^n$ 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

$$v = \begin{pmatrix} 1 & 3 & 0 & 4 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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.

1. Écrire une fonction `creux` d'argument une liste simple représentant un vecteur v qui renvoie un booléen indiquant si v est creux ou pas.

2. Écrire deux fonctions : `coder` d'argument un vecteur qui renvoie son codage "creux"; `decoder` de deux arguments C et n, qui restitue le vecteur de dimension n à partir de son codage "creux" C.

   On construit maintenant des fonctions adaptées pour le codage "creux".

3. Écrire une fonction `smul` de deux arguments C et a, où C est le codage "creux" d'un vecteur v et a un scalaire, qui renvoie le codage creux du vecteur av.

4. Écrire une fonction `pscal` 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.

5. Écrire une fonction `add` de deux arguments `C1` et `C2`, codages "creux" de deux vecteurs v_1 et v_2, qui renvoie le codage "creux" du vecteur $v_1 + v_2$. Cette somme est-elle toujours un vecteur creux ?

Exercice 12 On dit qu'un nombre entier est une puissance finale d'un nombre entier non nul a si l'écriture en base 10 de a^n se termine par n, par exemple : $2^{36} = 68719476736$ donc 36 est une puissance finale de 2.

1. Écrire une fonction booléenne `PF` de deux arguments a et n, qui teste si n est une puissance finale de a.

2. Écrire une fonction `LF` de deux arguments a et N, qui renvoie la liste des puissances finales de a inférieures ou égales à N. Tester `LF(2,1000)` et `LF(7,1000)`.

3. Écrire une fonction `LF2` de deux arguments A et n, qui renvoie la liste des entiers inférieurs ou égaux à A dont n est une puissance finale. Tester `LF2(100,36)` et `LF2(1000,13)`.

4. Écrire une fonction `FF` d'un argument entier non nul a, qui renvoie le premier nombre n puissance finale de a en limitant le temps de recherche à 10 secondes (on pourra utiliser la fonction `time` du module `time`). Tester `FF(8)` et `FF(10)`.

Exercice 13 Dans cet exercice, les nombres entiers sont représentés par leur écriture en base 10 et leur écriture en base 2, chacune de ces écritures étant une chaîne de caractères. Par exemple, le nombre qui s'écrit $'18'$ en base 10 s'écrit $'10010'$ en base 2.

1. Écrire une fonction `somchi` d'argument une chaîne de caractères représentant un nombre entier n en base 10 et renvoyant la somme de ses chiffres décimaux.

2. Trouver tous les nombres entiers inférieurs à 10 000 égaux à la puissance quatrième de la somme de leurs chiffres en base 10.

3. Écrire une fonction de conversion `DixVersDeux` ayant pour argument une chaîne de caractères représentant l'écriture d'un nombre entier en base 10 et renvoyant son écriture en base 2.

4. Écrire la fonction de conversion `DeuxVersDix`.

5. Trouver tous nombres entiers inférieurs à 10 000 égaux à la puissance quatrième de la somme des chiffres de leur écriture en base 2.

6. Modifier la fonction `DixVersDeux` en une fonction `DixVersBase` de telle manière que la chaîne de caractères renvoyée comme résultat soit écrite en base b. La base b de numération sera passée comme deuxième argument.
   Trouver tous les nombres entiers inférieurs à 10 000 égaux à la puissance quatrième de la somme des chiffres de leur écriture en base b pour toutes les bases de 3 à 9.

Exercice 14

1. On considère le code *Python* suivant :

```
def chiffres(n):
    if n == 0 :
        return [0]
    L = []
    while n != 0:
        L.append(n%10)
        n = n//10
    return L[::-1]
```

   Que fait la fonction `chiffres` ?

2. Soit n un entier naturel non nul possédant p chiffres (en base 10). Dans cet exercice, on dit que n est un nombre "narcissique" si la somme des puissances p-ième de ses chiffres vaut n.
   Montrer que 93084 est un nombre narcissique.

3. Écrire une fonction `narcisse` d'argument n qui renvoie un booléen indiquant si n est un nombre narcissique, ou pas.

4. Afficher tous les nombres narcissiques compris entre 0 et 10000.

5. Écrire une fonction `narcis_suiv` d'arguments n et N qui renvoie le premier entier narcissique plus grand que n, en limitant la recherche aux nombres inférieurs ou égaux à N.

6. Déterminer l'ensemble des nombres premiers et narcissiques compris entre 1 et 10000.

Exercice 15

1. Écrire une fonction `partage` de deux arguments, une liste L d'entiers et un entier a, renvoyant le couple $(L_{\text{inf}}, L_{\text{sup}})$, où L_{inf} (resp. L_{sup}) désigne la liste des éléments de L inférieurs ou égaux (resp. strictement supérieurs) à a.

2. À l'aide de la fonction précédente, écrire une fonction réalisant le tri rapide ("quicksort") d'une liste.

3. Modifier les fonction précédentes de manière à renvoyer, outre la liste triée, le nombre total de comparaisons effectuées au cours du tri.

4. Tester votre programme sur une liste composée de 40000 entiers aléatoirement choisis entre -999 et 999 (on pourra utiliser par exemple la fonction `randint` de la bibliothèque `random`).

Exercice 16

1. Écrire une fonction `C1` d'argument un entier naturel n qui renvoie la liste ordonnée de tous les carrés parfaits (d'entiers naturels) inférieurs ou égaux à n. Par exemple, `C1(10)` donne $[0, 1, 4, 9]$. Tester `C1` pour $n = 100$.

2. Écrire une fonction `C2` d'argument un entier naturel 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. Tester `C2` pour $n = 100$.

3. Écrire une fonction `decomp2` d'argument un entier naturel m qui renvoie la liste de tous les couples (p, q) d'entiers naturels tels que $m = p^2 + q^2$ et $p \leqslant q$.

4. Écrire une fonction `C3` d'argument un entier naturel qui renvoie la liste ordonnée de tous les entiers naturels inférieurs ou égaux à n qui s'écrivent comme somme de trois carrés d'entiers naturels. Tester `C3` pour $n = 100$.

5. Vérifier que les entiers inférieurs à 2017 qui ne peuvent pas s'écrire comme la somme de trois carrés d'entiers sont exactement ceux que peuvent s'écrire $4^k(8q + 7)$.

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