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 La suite $(K_n)_{n\in\mathbb{N}}$ d'entiers naturels est définie par :

- $K_0 = 1$
- $\forall n \in \mathbb{N}^*,\ K_n = \sum_{i=0}^{n-1} K_i K_{n-1-i}$

1. Écrire une fonction récursive `K` d'argument un entier n et renvoyant K_n.
2. Calculer `K(2)`, `K(5)`, `K(10)` et `K(15)`. Que pensez-vous de l'efficacité de la fonction `K` ?
3. Écrire une fonction `LK` non récursive d'argument un entier n et renvoyant le nombre L_n donné par
$$L_n = \frac{1}{n+1}\binom{2n}{n}.$$
On pourra utiliser `comb` *qui se trouve dans la sous-bibliothèque* `scipy.misc` *du langage Python.*
4. Afficher en les comparant les 10 premières valeurs de K_n et de L_n. Que peut-on conjecturer ?
5. On admet que la conjecture précédente est vérifiée pour tout entier n. Après avoir exprimé L_n en fonction de L_{n-1}, proposer une méthode permettant un calcul plus efficace de K_n.

Exercice 2

1. Écrire une fonction `binaire` d'argument un entier naturel n et qui renvoie la liste des chiffres, bit de poids fort en tête, de l'écriture en base 2 de n. Par exemple, `binaire(23)` renvoie `[1,0,1,1,1]`.
2. Écrire une fonction `nombreDeUns` d'argument un entier naturel n, qui renvoie le nombre de chiffres de 1 dans l'écriture binaire de n. Par exemple, `nombreDeUns(23)` renvoie 4.
3. Soit n un entier naturel. On dit que n est un 2-palindrome si sa représentation en base 2 est la même, qu'elle soit écrite de gauche à droite ou de droite à gauche. Par exemple, 9 est un 2-palindrome car 9 s'écrit 1001 en base 2.

   Écrire une fonction `palindrome` d'argument un entier naturel n qui renvoie un booléen indiquant si n est un 2-palindrome, ou pas.
4. Faire afficher tous les 2-palindromes inférieurs à 100.
5. Écrire une fonction `baseB` de deux arguments, un entier naturel n et un entier b compris entre 2 et 10, qui renvoie la liste des chiffres, chiffre de poids fort en tête, de l'écriture en base b de n.
6. Faire afficher les 10 plus petits entiers naturels non nuls qui sont à la fois des 4-palindromes et des 9-palindromes.

Exercice 3 Une matrice carrée 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 constante s.

1. Au brouillon, exprimer la constante s en fonction de n.

2. Créer une fonction `EstMagique`, d'argument une matrice T (carrée de taille n) et qui renvoie un booléen indiquant si T est magique ou pas.
   Tester cette fonction sur les matrices :

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

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 avec les nombres 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. Construire "à la main", selon cette méthode, une matrice magique d'ordre 3.
4. Écrire une fonction `Magique` d'argument un entier p qui renvoie la matrice magique de taille $2p+1$ créée à l'aide de la méthode précédente.

Exercice 4 On travaille avec des triplets (jour, mois, année) où jour et année sont des nombres entiers (avec la condition année $\geqslant 1582$) : date de mise en place du calendrier grégorien) et mois est une chaîne de caractères.

1. Créer une liste notée `MC` contenant les mois de l'année qui ont 30 jours et une liste notée `ML` contenant les mois de l'année qui ont 31 jours.
2. On rappelle qu'une année est bissextile lorsqu'elle est divisible par 4 mais pas par 100, ou bien lorsqu'elle est divisible par 400. Créer une fonction `estBissextile` d'un argument `an` qui renvoie `True` si l'année `an` est bissextile et `False` sinon.
   Créer une fonction `longueurmois` de deux arguments `ms` et `an` qui renvoie le nombre de jours du mois `ms` de l'année `an`.
3. Créer une fonction `valide` de trois arguments `jr`, `ms` et `an` qui renvoie `True` si le triplet `(jr,ms,an)` est valide et `False` sinon. Par exemple `valide(25,'janvier',1896)` devra renvoyer `True` alors que `valide(30,'février',1972)` devra renvoyer `False`.
4. Écrire une fonction `nab` de deux arguments `date1` et `date2` renvoie le nombre de "29 février" entre ces deux dates. `date1` et `date2` sont deux triplets sous la forme `(jr,ms,an)`.
5. Écrire une fonction `jours` de deux arguments `date1` et `date2` qui renvoie le nombre de jours séparant les deux dates.

Exercice 5

1. On considère un nombre n. Que donne la commande `list(str(n))` ?

2. Écrire une fonction nommée `somme` d'argument un nombre entier naturel et qui renvoie la somme de ses chiffres.

3. On considère qu'un nombre est "*adéquat*" si la somme de ses chiffres est un multiple de 10. Écrire une fonction nommée `adequat` d'argument un entier naturel n et qui renvoie un booléen indiquant si n est "*adéquat*", ou pas.

4. Écrire 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 $n = p$.

5. Tester cette fonction en demandant d'afficher `adequat(modification(n))` pour 10 valeurs aléatoires de n entre 10000 et 100000 (fonction `randint` du module `random`).

6. Tirer au hasard plusieurs milliers d'entiers entre 1 et 10000 et calculer la proportion de nombres *adéquats*. Ce résultat vous surprend-il ?

Exercice 6 Dans cet exercice, on manipule des suites (finies) d'entiers sous 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` d'argument une liste d'entiers et qui renvoie un booléen indiquant si cette liste est croissante ou pas. Cette fonction devra être de complexité linéaire dans le pire des cas.

2. Écrire de même des fonctions `estDecroissante` et `estMonotone` qui testent si une liste d'entiers est respectivement décroissante, monotone.

Soit la liste $L = [u_0, u_1, \cdots, u_{n-1}]$ de longueur n. On appelle tranche de L une liste de la forme $[u_i, u_{i+1}, \cdots, u_j]$ où $0 \leqslant i \leqslant j < n$.
On cherche une tranche de L croissante et de longueur maximale.
Par exemple, une tranche croissante de longueur maximale de $[0, 1, 0, 1, 2, 3, 4, 0, 1, 2]$ est $[0, 1, 2, 3, 4]$, correspondant aux indices $i = 2$ et $j = 6$.

3. Écrire une fonction `LC` de deux arguments, une liste `L` et un entier `p`, qui renvoie l'entier `d` tel que : la liste $[u_p, \cdots, u_{p+d}]$ est croissante et soit `p+d=n-1`, soit $u_{p+d} > u_{p+d+1}$.

4. Écrire une fonction `maxCroissante` d'argument une liste `L` qui renvoie la plus longue tranche croissante de `L`. S'il n'y a pas unicité, on renvoie la première trouvée.

Exercice 7 Soit n un entier naturel impair et non multiple de 5. On admet qu'un de ses multiples noté N s'écrit un base 10 avec seulement le chiffre 1 (de la forme $111\cdots111$). 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.

1. Écrire une fonction `QueDesUn` d'un argument n qui renvoie le premier multiple N de n ne s'écrivant qu'avec des 1, si n est impair non multiple de 5, et `'erreur'` sinon.

2. Pour n allant de 1 à 150, pour les n impairs et non multiples de 5 afficher les couples (n, N).

3. Écrire une fonction `Longueur(k)`, donnant le nombre de chiffres de l'entier naturel k. [Indication : la fonction `log` ne pouvant pas s'appliquer aux entiers longs, on pourra compter le nombre de divisions entières par 10 que l'on peut faire pour obtenir finalement 0.]

4. Pour n allant de 1 à 1000, chercher le(s) couple(s) (n, N) tel que le nombre de 1 de N soit le plus grand. (En cas d'égalité, donner toutes les solutions.)

Exercice 8 Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par :

$$u_0 = N \text{ (entier naturel non nul)} \quad \text{et} \quad \forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} u_n/2 \text{ si } u_n \text{ est pair} \\ 3u_n + 1 \text{ si } u_n \text{ est impair} \end{cases}$$

1. À l'aide notamment des instructions `plot` et `show` de la bibliothèque `mathplotlib.pyplot`, représenter graphiquement les 50 premiers termes de la suite (u_n) avec $N = 7$, puis $N = 18$.
2. On conjecture que pour tout entier naturel N non nul, il existe un plus petit entier n tel que $u_n = 1$. Que se passe-t-il si c'est le cas ?
   Si cet entier existe, on l'appelle "durée de vol" de la suite (u_n) de valeur initiale N.
3. Écrire une fonction `vol` 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.
4. Représenter la durée de vol en fonction du point de départ N, pour les valeurs de N inférieures à 1000.
5. Représenter la distribution des durées de vol à l'aide d'un histogramme (on pourra utiliser la commande `hist` de `matplotlib.pyplot`).

Exercice 9 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.

1. Écrire une fonction `LDP` d'argument un entier naturel n qui renvoie la liste de ses diviseurs propres. La tester pour $n = 100$.
2. Écrire une fonction `SDP` d'argument un entier naturel n qui renvoie la somme de ses diviseurs propres. La tester pour $n = 100$.
3. On dit qu'un entier naturel est **parfait** s'il est égal à la somme de ses diviseurs propres. Écrire une fonction `parfaits` d'argument un entier naturel K qui renvoie la liste des entiers p parfaits inférieurs ou égaux à K, après avoir affiché au fur et à mesure le message "p est parfait". La tester pour $K = 2000$.
4. On dit que deux entiers sont `amicaux` si chacun est égal à la somme des diviseurs propres de l'autre. Écrire une fonction `amicaux` d'argument un entier naturel K qui renvoie la liste de tous les couples (p, q) d'entiers amicaux tels que $p < q \leqslant K$, après avoir affiché les couples trouvés au fur et à mesure. Tester `amicaux` pour $K = 5000$.

Exercice 10 Dans la liste des entiers naturels non nuls, on barre un nombre sur 2 en commençant par barrer le deuxième :

$$1\ ,\ \cancel{2}\ ,\ 3\ ,\ \cancel{4}\ ,\ 5\ ,\ \cancel{6}\ ,\ 7\ ,\ \cancel{8}\ ,\ 9\ ,\ \cancel{10}\ ,\ 11\ ,\ \cancel{12}\ ,\ 13\ ,\ \cdots$$

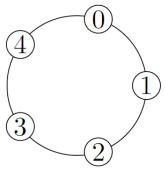
Puis dans la liste restante, on barre une nombre sur 3 en commençant par barrer le troisième :

$$1\ ,\ 3\ ,\ \cancel{5}\ ,\ 7\ ,\ 9\ ,\ \cancel{11}\ ,\ 13\ ,\ \cdots$$

puis on barre un nombre sur 4, un nombre sur 5, etc.
Et ceci à l'infini pour obtenir "*la liste des nombres de type J*".

1. Écrire une fonction `enlever` de deux arguments, une liste L et un entier naturel i, qui renvoie une liste S construite en ne gardant dans la liste L que les éléments dont le rang n'est pas multiple de i. Par exemple, `enlever([1,3,5,7,9,11,13],3)` doit donner `[1,3,7,9,13]`.
2. Écrire une suite d'instructions donnant la liste des nombres de type J inférieurs ou égaux à 100.
3. Écrire une fonction `LJ` d'argument n renvoyant la liste des nombres de type J inférieurs ou égaux à n.
4. Écrire une fonction `U` d'argument n renvoyant u_n, le nombre de nombres de type J inférieurs ou égaux à n.
5. Vers quelle limite ℓ semble tendre $4n/u_n^2$ quand n tend vers l'infini ?
6. Déterminer le premier n pour lequel la différence en valeur absolue entre $4n/u_n^2$ et ℓ est inférieure à 10^{-3}.

Exercice 11 n personnes numérotées de 0 à $(n-1)$ se mettent en cercle, comme le montre la figure suivante pour $n = 5$:



En commençant par la personne numéro 1 et en tournant dans le sens des numéros croissants (sens horaire sur la figure), on retire une personne sur deux, en ne prenant en compte que les personnes restant dans le cercle. Par exemple, pour $n = 5$, on retirera successivement les personnes 1, 3, 0 puis 4; il restera la personne 2.

Pour simuler cette procédure d'élimination progressive, on décrit le cercle par une liste de n booléens : la valeur numéro i est `True` si la personne i est éliminée, et `False` si elle est encore dans le cercle.

Au départ, la liste ne contient que des `False` puis ses valeurs passent progressivement à `True`, jusqu'à ce qu'il ne reste plus qu'un seul `False`.
Ainsi, pour $n = 5$, la liste passe par les états successifs :

Liste de booléens `E`	Rang `p` du dernier éliminé
`[ False , True , False , False , False ]`	1
`[ False , True , False , True , False ]`	3
`[ True , True , False , True , False ]`	0
`[ True , True , False , True , True ]`	4

Il reste 2

1. Écrire une fonction `suivant` de deux arguments, une liste `E` de n booléens et un entier `p` entre 0 et $(n-1)$ qui renvoie la position `q` du premier `False` rencontré en partant de la position juste après la position `p`, en parcourant la liste de façon circulaire.
   Par exemple :

   `suivant([True,True,False,True,False],0)` donne 2;
   `suivant([True,True,False,True,False],2)` donne 4;
   `suivant([True,True,False,True,False],4)` donne 2.

2. Se servir de la fonction `suivant` pour simuler le cas $n = 5$ décrit dans le tableau ci-dessus.

3. Écrire une fonction `reste` d'argument un entier naturel non nul n qui renvoie le numéro de la dernière personne restante parmi n personnes. Tester `reste` pour $n = 16$, $n = 31$, $n = 65$.

4. Pour $2 \leqslant n \leqslant 140$, afficher n suivi du numéro du dernier restant.
En observant le résultat, que peut-on conjecturer sur le calcul du dernier restant ?

5. On suppose que la conjecture est exacte, deviner sans utiliser le programme Python quel sera le numéro du dernier restant pour les valeurs de n : 256, 513, 1023, 1041.

6. Écrire une fonction `reste2` mettant en oeuvre cette conjecture et comparer les résultats pour $n = 5104$.