— Préparation Python —

Exercice 1. — Écrire une fonction `Emoy(L)` qui étant donné une liste `L` d'entiers, revoie le nombre d'éléments qui sont strictement inférieurs à la moyenne des éléments de `L`.

Exercice 2. — Écrire une fonction `sigma(L)` qui prend en entrée une liste d'entiers

$$\texttt{L=[}\mathtt{a}_0\texttt{,}\mathtt{a}_1\texttt{,}\ldots\texttt{,}\mathtt{a}_{n-1}\texttt{]}$$

et qui renvoie l'entier $\sum_{i=1}^{n-1} \prod_{j=i}^{n-1} \mathtt{a}_j$. Pour obtenir tous les points, il faudra proposer une fonction utilisant au plus n multiplications.

Exercice 3. — On considère l'application σ qui à un entier associe la somme des cubes des ses chiffres dans son écriture en base 10. Par exemple, on aura $\sigma(124) = 1^3 + 2^3 + 4^3 = 1 + 8 + 64 = 73$. On se propose d'étudier lorsque a est un entier multiple de 3 non nul, la suite récurrente définie par

$$\begin{cases} u_0 = a, \\ u_{n+1} = \sigma(u_n) \quad \text{si } n \in \mathbb{N}. \end{cases}$$

1°) Écrire une fonction `sigma` qui prend en argument un entier naturel et qui retourne la somme des cubes des ses chiffres dans son écriture en base 10.

2°) On pose $u_0 = 24258$. Afficher la liste `L` des vingt premiers termes de la suite. Faire de même lorsque l'on prend $u_0 = 999\,999\,999$. Est ce qu'ils ont un diviseur commun ? Que se passe-t-il à partir d'une certain rang ?

3°) Déterminer la liste des entiers i de l'intervalle $[\![1, 9999]\!]$ vérifiant $\sigma(i) = i$.

4°) Montrer que pour tout $k \in [\![1, 9999]\!]$ où $k \equiv 0 \pmod 3$ on a $\sigma^{13}(k) = 153$.

5°) Conjecturer la nature de la suite. Argumenter.

Exercice 4. — Écrire une fonction `hyperfactorielle(n)` qui prend en argument un entier naturel `n` non nul et qui retourne le produit $\prod_{k=1}^{n} k^k$.

Exercice 5. — Écrire une fonction `prixPhotocopies(n)` qui affiche le prix de `n` photocopies sachant que le reprographe facture 0.10 euro les dix premières photocopies, 0.09 euro les vingt suivantes et 0.08 euro au-delà.

Exercice 6. —

1°) Étant donné un entier naturel n, construire une procédure `prodchiffre` prenant en argument un entier naturel n et qui retourne le produit de ses chiffres dans son écriture en base 10. Par exemple,

`prodchiffre(355)` $= 3 \times 5 \times 5 = 75$, `prodchiffre(75)` $= 7 \times 5 = 35$, `prodchiffre(5)` $= 5$.

2°) On note alors u_n l'entier compris entre 0 et 9 et obtenu en itérant la fonction `prodchiffre` sur l'entier n. Construire une procédure `u` prenant en argument un entier naturel n et qui retourne le chiffre recherché. Par exemple, on a

$$\begin{array}{ll} 355 \to 75 \to 35 \to 15 \to 5 & \text{donne } u_{355} = 5 \\ 1789 \to 504 \to 0 & \text{donne } u_{1789} = 0 \\ 999 \to 729 \to 126 \to 12 \to 2 & \text{donne } u_{999} = 2 \end{array}$$

3°) La persistance de $n \in \mathbb{N}$, notée `per(n)`, est le nombre minimal d'itérations de `prodchiffre` nécessaire pour transformer n en u_n. Par exemple,

`per(5)=0`, `per(15)=1`, `per(355)=4`.

Construire une procédure `per` retournant cette persistance.

4°) Construire la liste `plus_grande_per` des entiers de l'intervalle $[\![0, 1\,000\,000]\!]$ ayant la plus grande persistance. Donner le nombre d'éléments de cette liste.

Exercice 7. —

1°) Écrire une fonction `Qprime(n)` qui retourne `True` si `n` est un nombre premier et `False` sinon.

2°) Donner la liste des 50 premiers nombres premiers.

3°) Donner la liste des nombres premiers de $[\![1, 1000]\!]$. On pourra proposer plusieurs méthodes.

4°) Donner la liste des nombres premiers de $[\![1, 1000]\!]$ qui sont congrus à 1 modulo 4. On pourra proposer plusieurs méthodes.

5°) Donner la liste de vingt entiers consécutifs $[\![n, n+19]\!]$ avec n le plus petit possible ne contenant aucun nombre premier.

Exercice 8. —

1°) On écrit sur une feuille tous les entiers entre 1 et 2014 (c'est une très grande feuille). Combien de fois apparaît le chiffre 1 ? On pourra envisager plusieurs méthodes.

2°) Trouver maintenant le plus petit entier naturel n tel qu'en écrivant tous les entiers entre 1 et n on aura écrit 2014 fois le chiffre 1.

Exercice 9. — Construire une procédure `zerosfact(n)` prenant en paramètre un entier naturel `n` et qui compte le nombre de zéros qui termine l'écriture de `n`!. On donnera au passage le nombre de zéros qui termine l'écriture de 2014!.

Exercice 10. — Nous allons voir dans cet exercice les matrices comme une liste de listes. Par exemple la matrice

$$\mathrm{M} = \begin{bmatrix} 5 & 3 \\ 0 & 7 \\ 4 & 1 \end{bmatrix}$$

se représente par `M=[[5,3],[0,7],[4,1]]`.

On dit qu'une matrice $A = \left[a_{i,j}\right]_{(i,j)\in[\![1,n]\!]^2}$ de $\mathfrak{M}_n(\mathbb{N})$ est pseudo-magique lorsqu'il existe $\alpha \in \mathbb{N}$ tel que

$$\forall i \in [\![1,n]\!], \qquad \sum_{k=1}^{n} a_{ik} = \sum_{k=1}^{n} a_{ki} = \alpha$$

Autrement dit, la somme des coefficients des lignes et des colonnes de la matrice A est constante. Par exemple, la matrice suivante

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix}$$

est pseudo-magique. Écrire une procédure `Qmagik(M)` qui renvoie `True` lorsque `M` est pseudo-magique et `False` dans le cas contraire. Il faudra commencer par vérifier que la matrice `M` est bien carrée.

Exercice 11. — On considère la suite $(v_n)_{n\in\mathbb{N}}$ dfinie par $v_0 = 0$, $v_1 = 1$, et $v_{n+2} = 2v_n - v_{n+1}$ pour tout $n \in \mathbb{N}$.

1°) Écrire une fonction `v(n)` calculant v_n. Expliquer le choix fait dans la boucle.

2°) Tester cette fonction sur $n = 3$. Le résultat retourné est-t-il le bon ?

3°) Donner les instructions permettant d'écrire dans un fichier nommé `psi.txt` les lignes de la forme des tuples (n, v_n), pour $n \in [\![0, 1000]\!]$. Les premières lignes sont donc :
$(0, 0)$
$(1, 1)$
$(2, -1)$
$(3, 3)$

4°) Comment, à partir de ce fichier (qu'on va donc lire), calculer la somme des $n \in [\![0, 1000]\!]$ tels que v_n est divisible par 113 ?

Exercice 12. — On définit la fonction suivante qui s'applique à des entiers naturels.

Python

```python
def mystere(n):
    compteur=0
    while n!=0:
        if n%10==7:
            compteur+=1
        n=n//10
    return compteur
```

1°) Qu'obtient-on à l'appel de la procédure `mystere` pour l'entier 1721771 ? Remplir le tableau de l'évolution des variables à l'appel de la procédure :

`compteur`	0							
`n`	1721771							

Entrée dans la boucle

De manière générale, que renvoie cette procédure à l'appel d'un entier naturel `n` ? On ne demande pas commenter l'algorithme ligne à ligne mais dire simplement ce qu'elle renvoie.

2°) Modifier la procédure ci-dessus pour créer une procédure `vingtcinq(n)` qui compte le nombre de motif « 25 » dans un entier naturel `n` donné en argument. Par exemple,

`vingtcinq(25)=1`, `vingtcinq(72582525)=3`, `vingtcinq(7265)=0`.

Exercice 13. — On rappelle qu'un entier naturel n est impair lorsqu'il existe $k \in \mathbb{N}$ tel que $n = 2k + 1$ c'est-à-dire lorsque son reste dans la division euclidienne par 2 vaut 1. Attention, on vous demande de faire les questions dans l'ordre.

1°) Construire une procédure `QImpair(L)` prenant comme paramètre une liste non vide d'entiers naturels et renvoyant `True` si l'un des termes de la liste est impair et `False` sinon.

2°) Construire une procédure `FirstImpair(L)` prenant comme paramètre une liste `L` non vide d'entiers naturels et qui renvoie le premier entier impair que l'on rencontre dans la liste s'il y en a au moins un et `None` dans le cas contraire.

3°) Construire une procédure `LastImpair(L)` prenant comme paramètre une liste `L` non vide d'entiers naturels et renvoyant le dernier entier impair que l'on rencontre dans la liste s'il y en a au moins un et `None` dans le cas contraire.

4°) Construire une procédure `SommeImpair(L)` prenant comme paramètre une liste `L` non vide d'entiers naturels et renvoyant la somme de tous les entiers impairs de la liste donnée en argument.

5°) Construire une procédure `ExtractionImpair(L)` prenant comme paramètre une liste `L` non vide d'entiers naturels et qui renvoie la liste des entiers impairs contenus dans la liste `L` dans leur ordre d'apparition.

Exercice 14. —

L'objectif de cet exercice est de construire une procédure `nombreZerosMax` donnant le nombre maximum de zéros consécutifs dans une liste de bits. Par exemple, pour le tableau ci-dessous

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L[i]	0	1	1	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0

la procédure `nombreZerosMax` devra renvoyer 5.

1°) Écrire une fonction `NombreZeros(i,L)`, prenant en paramètre une liste `L` non vide et un indice `i` compris entre 0 et `len(L)`-1 et

- qui retourne 0 si `L[i]`$=1$
- qui retourne le nombre de zéros consécutifs qu'il y a dans `L` à partir de `L[i]` (inclus) dans le cas où `L[i]`$=0$.

Par exemple,

`NombreZeros(0,L)` $= 1$, `NombreZeros(5,L)` $= 0$, `NombreZeros(6,L)` $= 3$.

2°) Comment obtenir le nombre maximal de zéros contigus d'une liste `L` connaissant la liste des `NombreZeros(i,L)` lorsque `i`$\in [\![0, n-1]\!]$ où n désigne `len(L)`.

3°) En déduire une procédure `nombreZerosMax(L)`, qui renvoie le nombre maximal de zéros consécutifs d'une liste `L` non vide, qui utilise la fonction `NombreZeros`. Pour cette procédure je vous demande de ne pas utiliser la fonction `max` de `python`.

4°) Proposer une procédure `nombreZerosMax` plus efficace.

Exercice 15. — L'objectif de cet exercice et de travailler avec les bibliothèques `scipy` et `numpy`. Les trois questions quisuivent sont indépendantes.

1°) a) Donner une approximation de

$$\int_0^1 \cos(\sqrt{t})\,\mathrm{d}t$$

à l'aide de la fonction `quad` de la bibliothèque `scipy.integrate`.

b) Faire de même en implémentant la méthode des trapèzes.

2°) Donner une approximation de l'unique réel positif solution de l'équation $x^2 + \sqrt{x} = 1$.

3°) Donner une approximation de l'unique réel positif t tel que

$$\int_1^t (1 + \sqrt{x} + \cos(x))\,\mathrm{d}x = 10.$$

Exercice 16. — On considère dans le plan le triangle équilatéral T dont les sommets sont les points ayant comme affixes les racines cubiques de l'unité. Nous allons construire la marche aléatoire de l'origine à l'intérieur du triangle selon le protocole suivant. Le point de départ, c'est l'origine $\mathrm{M}_0(0,0)$ centre de gravité du triangle T. Le point suivant M_1, se situe au milieu du segment dont les extrémités sont M_0 et l'un des sommets de T pris de façon aléatoire puis on réitère le procédé pour construire la suite de points (M_n).

Pour le coté aléatoire, les graphiques et les fonctions usuelles, nous aurons besoin de quelques bibliothèques de Python :

Python

```python
import numpy as np
import matplotlib.pyplot as plt
from random import *
```

1°) Construire à l'aide de la fonction `choice`[1] une fonction `sommet_alea()` qui renvoie de façon aléatoire l'un des trois sommets du triangle T dont les coordonnées sont

$$\left[-\frac{1}{2}, \frac{\sqrt{3}}{2}\right], \qquad \left[-\frac{1}{2}, \frac{-\sqrt{3}}{2}\right], \qquad [1,0]$$

2°) Construire une fonction `liste_points(n)` qui retourne la liste `[abscisses,ordonnees]` contenant la liste `abscisses` des abcsisses des points $(\mathrm{M}_i)_{i\in[\![0,n]\!]}$ et la liste `ordonnees` des ordonnées des points $(\mathrm{M}_i)_{i\in[\![0,n]\!]}$.

3°) À l'appelle de la fonction

`plt.plot(abscisses,ordonnees,'x',ms=2.,color='purple')`

On obtient les points de coordonnées `[`$\mathtt{x}_i$`,`$\mathtt{y}_i$`]` où $\mathtt{x}_i$ et $\mathtt{y}_i$ sont respectivement les éléments d'indice `i` dans les deux listes `abscisses` et `ordonnees`. Représenter les points de la marche aléatoire en prenant une valeur de n significative.

1. La fonction `choice` appliquée sur une liste renvoie un élément aléatoire de la liste.

4°) Faire de même en faisant varier la configuration de départ : carré, pentagone, etc.