Exercices hebdomadaires

Chaque exercice est à rédiger sur feuille et à me rendre en main propre en début de chaque séance ou à déposer dans mon casier le jour indiqué.

Exercice 1

On considère les fonctions f_1 et f_2 définies sur $[0, 2]$ par :

$$f_1(x) = \begin{cases} x & \text{si } 0 \leqslant x < 1 \\ 1 & \text{sinon.} \end{cases}$$

et

$$f_2(x) = \sin(x) + 0,1$$

1. Définir les deux fonctions `f1` et `f2` dans Python en utilisant le module `math`.
2. Proposer un code qui trace les courbes représentatives des fonctions sur l'intervalle $[0, 2]$ avec un pas de 0,05. Le graphique doit avoir les caractéristiques suivantes :
   - Utilisation de listes
   - Courbe de `f1` : épaisseur du trait égale à 3, couleur bleue.
   - Courbe de `f2` : points non reliés représentés par $*$, couleur rouge.
   - Légender les figures
   - Afficher `x` pour l'axe des abscisses et `y` pour l'axe des ordonnées.
   - Afficher le titre `'Tracé de fonctions'`
   - Axe des x compris entre 0 et 2, axe des ordonnées compris entre 0 et 1,5.

Exercice 2

Soit `L` une liste de longueur n dont les termes valent 0 ou 1. On cherche le nombre maximal de 0 contigus dans `L` (c'est-à-dire figurant dans des cases consécutives). Par exemple, le nombre maximal de zéros contigus dans la liste suivante est 4 :

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`L[i]`	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0

1. Écrire une fonction `nombreZeros(L, i)` qui admet pour argument une liste `L` de longueur n, un indice `i` compris entre 0 et $n-1$ et qui renvoie :
   - 0, si `L[i]`$= 1$
   - le nombre de zéros consécutifs dans `L` à partir de `i` inclus, si `L[i]` $= 0$.
2. Comment obtenir le nombre maximal de zéros contigus d'une liste `L` connaissant la liste des `nombresZeros(L, i)` pour $0 \leqslant i \leqslant n-1$? En déduire une fonction `nombreZerosMax(L)` renvoyant le nombre maximal de 0 contigus d'une liste `L` non vide.
3. Quelle est la complexité de la fonction `nombreZerosMax` ?

Exercice 3

Écrire une fonction `recherche_2_max` qui admet comme argument une liste `L`. Cette fonction renvoie le maximum et le second maximum de cette liste. Exemple : soit `L1 = [2, 3, 1, 7, 3]` et `L2 = [2, 3, 1, 7, 7]`. `recherche_2_max(L1)` renvoie `7, 3` et `recherche_2_max(L2)` renvoie `7, 7`.

Exercice 4

1. Pour calculer le pgcd de 3705 et 513, on peut passer en revue tous les entiers 1, 2, 3, ..., 512, 513 puis renvoyer parmi ces entiers le dernier qui divise à la fois 3705 et 513. Il sera alors le plus grand des diviseurs communs à 3705 et 513. Écrire une fonction `pgcd` qui renvoie le pgcd de deux entiers naturels non nuls, selon la méthode décrite ci-dessus.
2. Écrire une fonction **récursive** `euclide_rec` permettant de calculer le pgcd de deux entiers naturels selon l'algorithme d'Euclide :
   - pour $b = 0$: $pgcd(a, 0) = a$
   - pour $b \neq 0$, on note r le reste de la division euclidienne de a par b, alors $\text{pgcd}(a, b) = \text{pgcd}(b, r)$

Exercice 5

On dispose des pièces de monnaies de valeurs entières suivantes : `S = [1, 2, 5, 10, 20, 50, 100]`. On suppose que la liste `S` est triée par ordre croissant de valeurs. On cherche à rendre une certaine somme entière `x` en utilisant le moins de pièces, qui peuvent être identiques.

1. On utilise la méthode la plus intuitive qui consiste à commencer par rendre la plus grande pièce possible. Pour `x=11`, on commence par rendre la pièce de 10. On appelle `L[x]` le nombre de pièce nécessaires pour rendre la somme `x`, on peut écrire la récurrence suivante :
   - `L[0]` $= 0$
   - Si $x \geqslant 1$: `L[x] = 1 + L[x - S[i]]` avec `i` le grand indice tel que `S[i]` $\leqslant$ x.

   Écrire une fonction récursive `rendu` qui admet comme arguments une liste `S` et un entier `x`. La fonction renvoie le nombre de pièce nécessaire pour rendre la somme `x` en utilisant la relation écrite ci-dessus.
2. Comment s'appelle la méthode utilisée dans l'algorithme précédent ? Est-ce que `rendu([1, 4, 6], 8)` renvoie la solution optimale ?

Exercice 6

1. Écrire un programme Python permettant de créer le fichier `'droite.txt'` contenant les éléments suivants :

   10 # nombre de points du fichier, ce commentaire n'est pas à écrire !
   25 ;10.9 # abscisse et ordonnée du premier point, ce commentaire n'est pas à écrire !
   20 ;9.3
   15 ;8.2
   12 ;7.5
   9 ;6.2
   6 ;5.8
   3 ;4.2
   0 ;3.9
   -3 ;2.8
   -6 ;2

2. Écrire un programme Python permettant d'ouvrir le fichier `'droite.txt'` et de récupérer les abscisses et les ordonnées dans deux listes.

Exercice 7

Les instructions suivantes permettent de récupérer dans la liste `L` l'ensemble des pixels d'une image en niveau de gris. Cette liste contient à la suite les pixels de la première ligne de l'image, de la deuxième ligne, ⋯ Les valeurs des pixels sont des entiers qui varient entre 0 (noir) et 255 (blanc).

```
from PIL import Image
img = Image.open('photo.png')
p, n = img.size                # n : nombre de lignes (hauteur)
L = list(img.getdata()         # p : nombre de colonnes (largeur) }
```

1. On choisit, ici, d'utiliser les listes de listes pour représenter l'image. Écrire une fonction qui admet comme argument l'image en niveau de gris `L` et qui renvoie une liste de liste. La première sous-liste contient la première ligne de l'image, la deuxième sous-liste contient la deuxième ligne ⋯
2. Écrire une fonction `negatif(M)` qui prend en argument l'image `M` et qui renvoie une image de même taille représentant le négatif de `M`. Attention, cette fonction ne doit pas modifier `M`.

Exercice 8

On souhaite réaliser un lissage de l'image `M` de l'exercice précédent. On considère la matrice `A` suivante :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Le traitement suivant est appliqué à la matrice `M`. Pour calculer la nouvelle valeur du pixel de l'image traitée :

- On multiplie son ancienne valeur $M_{i,j}$ par la valeur centrale de la matrice `A`.
- On additionne les valeurs des pixels adjacents au pixel traité multipliée par les valeurs des éléments adjacents à l'élément central de la matrice `A` : $A_{0,0} \times M_{i-1,j-1} + A_{1,0} \times M_{i,j-1} + A_{2,0} \times M_{i+1,j} + \cdots$

Écrire une fonction qui `filtre(M, A)` qui admet comme arguments l'image `M` à filtrer et le filtre `A` à lui appliquer. `A` est une matrice (liste de liste) de taille 3×3.

Exercice 9

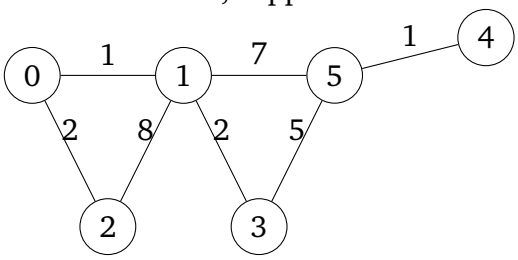
Le tri par comptage est un algorithme de tri d'entiers. On considère des entiers de 0 à p dans une liste `L` contenant n éléments. L'algorithme compte le nombre d'occurrences de chaque entiers. Mise en place de l'algorithme :

- On définit une liste vide `L_tri`.
- On définit une liste `H` de $p+1$ valeurs initialisées à 0. (Attention : p n'est pas connu *a priori*.
- On parcourt la liste `L`, on compte le nombre de fois qu'apparaît `L[i]` et on incrémente `H[L[i]]` de 1 à chaque fois.
- On parcourt la liste `H` pour construire au fur et à mesure la liste triée `L_tri`.

1. Écrire une fonction `tri_comptage` réalisant cette opération.
2. Évaluer la complexité de cet algorithme de tri.
3. Proposer une modification en tenant compte du minimum et du maximum de la liste `L`.

Exercice 10

On considère le graphe $G = (S, A)$ non orienté suivant. Le nombre situé sur l'arrête joignant deux sommets est leur distance, supposée entière.

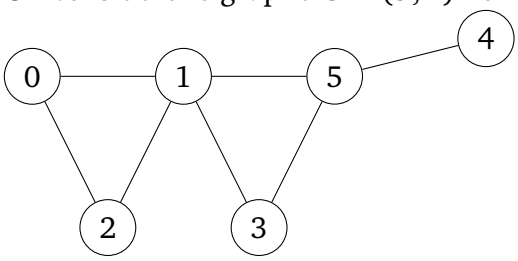


On utilise les listes de listes pour représenter les matrices dans Python.

1. Construire la matrice d'adjence `M` du graphe `G`. On convient que lorsque deux sommets ne sont pas reliés la distance les séparants est `inf`. On définira `inf = 1e20`.
2. Écrire une fonction `voisins(M, i)`, d'argument `M` une matrice d'adjacence et `i` un sommet, et qui renvoie la liste des voisins du sommet `i`.
3. Écrire une fonction `longueur(M, L)`, d'arguments une matrice d'adjacence `M`, une liste `L` de sommets de `G`, renvoyant la longueur du trajet décrit par la liste `L`. Si le trajet n'est pas possible, la fonction renverra -1.

Exercice 11

On considère le graphe $G = (S, A)$ non orienté suivant.



1. Construire la matrice d'adjence `M` du graphe `G`. Si deux sommets différents i et j sont reliés alors $M_{i,j} = 1$, sinon $M_{i,j} = 0$.
2. Écrire une fonction `test_graphe_connexe_profondeur(M)` qui admet la matrice d'adjence `M` en argument et qui renvoie `True` si graphe est connexe et `False` sinon en utilisant **le parcours en profondeur**.

Exercice 12

En entrant dans le hall d'une gare, vous découvrez un petit problème : le sol est en cours de rénovation. Vous ne pouvez même pas atteindre les guichets tant qu'ils n'ont pas fini d'installer le nouveau carrelage. Les carreaux sont tous hexagonaux ; ils doivent être disposés dans une grille hexagonale avec un motif de couleur très spécifique. N'étant pas d'humeur à attendre, vous proposez votre aide pour trouver le motif. Les carreaux sont blancs d'un côté et noirs de l'autre. Initialement, ils sont tous avec le côté blanc vers le haut. Le hall d'entrée est assez grand pour accueillir n'importe quel motif. Un membre de l'équipe de rénovation vous donne la liste des carreaux qui doivent être retournés (la liste de liste `L`). Chaque élément de la liste `L` identifie un carreau unique qui doit être retourné en donnant une série d'étapes à partir d'un carreau de référence situé au centre de la pièce (chaque ligne part du même carreau de référence). Comme les carreaux sont hexagonaux, ils ont tous six voisins : est, sud-est, sud-ouest, ouest, nord-ouest et nord-est. Ces directions sont indiquées dans chaque sous-liste, respectivement par `'e'`, `'se'`, `'so'`, `'o'`, `'no'` et `'ne'`. Un carreau est identifié par une série de ces directions ; par exemple, `['e', 'se', 'ne', 'e']` identifie le carreau sur lequel vous arrivez en vous déplaçant d'un carreau vers l'est, d'un carreau vers le sud-est, d'un carreau vers le nord-est et d'un carreau vers l'est.
Chaque fois qu'un carreau est identifié, il passe du blanc au noir ou du noir au blanc. Les carreaux peuvent être retournés plus d'une fois. Par exemple, une instruction comme `['no' , 'o', 'so', 'e', 'e']` retourne le carreau de départ.
Écrire un algorithme qui, à partir de la liste de liste `L`, détermine le nombre de carreaux noir visibles.

Exercice 13

Les instructions suivantes permettent d'ouvrir une image couleur et de la stocker dans la variable `image`. Il s'agit d'un tableau `numpy` à 3 dimensions, tel que `image[i, j, 0]` donne l'intensité (entier compris entre 0 et 255) de la composante rouge du pixel de coordonnée (i, j), `image[i, j, 1]` donne l'intensité de la composante verte et `image[i, j, 2]` donne l'intensité de la composante bleue.

```
import numpy as np
import matplotlib.image as img

image = img.imread('photo.png')
```

On souhaite augmenter le contraste de l'image, pour cela on applique à chaque pixel la fonction f définie par : $f(x) = 123\left(1 + \sin\left(\frac{\pi}{123}(x - 123)\right)\right)$.
Écrire une fonction qui prend en argument l'image de départ et qui renvoie une image plus contrastée. Attention, votre fonction ne doit pas modifier l'image initiale mais en créer une nouvelle.

Exercice 14

LVH Enterprises achète de longues tiges d'acier et les coupe en tiges plus courtes ensuite. Chaque coupe est gratuite. La direction de LVH Enterprises souhaite connaître la meilleure façon de couper les tiges. Nous supposons que nous connaissons, pour $i \in [1; n]$, le prix p_i en euros que LVH Enterprises demande pour une tige de longueur i. Les longueurs des tiges sont toujours un nombre entier. Attention, la loi du marché est telle que le prix n'est pas proportionnel à la longueur. Exemple :

i	1	2	3	4	5	6	7	8
p_i	1	5	6	8	10	12	20	22

Étant donné une tige de longueur n et un tableau des prix p_i, on veut déterminer la recette maximale r_n obtenue en découpant la tige et en vendant les morceaux.

1. Justifier que $r_n = \max_{1 \leqslant i \leqslant n}(p_i + r_{n-i})$
2. Écrire une fonction `r(p,n)` qui prend en argument une liste `p` de prix et la taille n de la tige et qui renvoie la recette maximale. On utilisera une méthode descendante par mémoïsation.

Exercice 15

On termine l'exercice précédent...
Écrire une fonction `decoupe(p, n)` qui prend en argument une liste `p` de prix et la taille n de la tige et qui renvoie la liste des découpes à effectuer.

Exercice 16

On considère le procédé suivant :
Soit $m \in \mathbb{N}$ que l'on suppose écrit en base 10. On enlève à l'écriture décimale de m le chiffre des unités et on le retranche deux fois au nombre ainsi obtenu. On note m' le résultat de cette opération.
Par exemple :

$$31976 \to 3197 - 2 \times 6 = 3185 \to 318 - 2 \times 5 = 308 \to 30 - 2 \times 8 = 14$$

1. Écrire une fonction Python qui réalise l'opération $m \to m'$ ainsi décrite. On la notera f.
2. Prouver que le mini-programme qui suit termine lors d'un appel avec m entier.

```
def critere7(m) :
  while m > 99 :
    m = f(m)
  return m
```

3. *facultatif :* Montrer que m est divisible par 7 si et seulement `critere7(m)` est divisible par 7.

Exercice 17

1. Soit a et b deux entiers naturels, q et r le reste dans la division euclidienne de a par b caractérisés par $a = bq + r$ et $0 \leqslant r < b$. Justifier que (a, b) et b, r ont les mêmes diviseurs.
2. On définir un algorithme de la façon suivante :
   Entrée : a et b entiers positifs ou nuls ; $a \neq 0$ ou $b \neq 0$
   Initialisation : $r_0 = a$, $r_1 = b$, $i = 1$.
   Tant que $r_1 \neq 0$:
   . Calculer (q, r) le couple formé du quotient et du reste de la division euclidienne de r_0 par r_1.
   . Faire $r_0 = r_1$, $r_1 = r$.
   (a) Montrer que ce programme termine.
   (b) Déterminer un invariant de boucle.
   (c) Justifier que, lorsque la boucle termine, $r_0 = \text{pgcd}(a, b)$.

Exercice 18

On considère la suite $(s_n)_n$ des nombres rationnels définie par :

$$s_n = \sum_{k=1}^{n} \frac{1}{k}$$

1. Écrire un script, de complexité linéaire, qui :
   - Définit le flottant `x = 12.3`
   - Détermine le premier entier N tel que $s_N > x$ et affiche le résultat.
2. Montrer par récurrence sur n que l'algorithme terminerait en donnant le résultat attendu **si on pouvait calculer en précision inifinie,** sachant que dans le cours de maths on démontre que $\lim\limits_{n\to\infty} s_n = +\infty$.
3. Que se passe-t-il en pratique ?

Exercice 19

Que pensez-vous de cette façon de programmer l'exponentiation rapide :

```
def expo(a, n):
  if n == 0 :
    return 1
  else :
    q, r = divmod(n, 2)
    p = expo(a, q)*expo(a, q)
    if r == 1 :
      p = p*a
    return p
```

Exercice 20

Un nombre heureux est un nombre entier qui, lorsqu'on ajoute les carrés de chacun de ses chiffres, puis les carrés des chiffres de ce résultat et ainsi de suite jusqu'à l'obtention d'un nombre à un seul chiffre égal à 1.
Écrire une fonction `heureux(nb)` qui permet de déterminer si un nombre entier `nb` est heureux ou non.

Exercice 21

Étant donné un tableau rectangulaire `A` contenant des 0 et des 1, on souhaite déterminer le plus grand carré constitué uniquement de 1 qu'on peut trouver à l'intérieur du tableau. Le tableau `A` sera donné sous forme d'un array `numpy`. Nous allons utiliser un dictionnaire `D` tel que `D[(i,j)]` est la longueur du côté du plus grand carré ne contenant que des 1 et dont le coin inférieur droit est la case (i, j).

1. Déterminer les "cas de base", i.e. pour quels pixels (i, j) la valeur de $D[(i, j)]$ est simple à préciser.
2. Dans le cas général, exprimer $D[(i, j)]$ en fonction de $D[(i-1, j-1)]$,$D[(i-1, j)]$et $D[(i, j-1)]$. *Aidez-vous d'un dessin.*
3. Coder une fonction `pgc1_memo(A)` en programmation dynamique par mémoïsation qui renvoie le dictionnaire D contenant les valeurs $D[(i, j)]$ pour chaque pixel (i, j).

Exercice 22

On continue...
Coder une fonction `emplacement(A)` donnant en sortie une liste `[n,(i0 ,j0 )]` correspondant à la taille n "du" plus grand carré et aux coordonnées `(x0 ,y0 )` du pixel lié à ce plus grand carré.

Exercice 23

Écrire le code qui permet d'afficher les représentations graphiques de la famille de fonctions f_n telle que $f_n(x) = \frac{1-nx}{1+x}$ pour n allant de -3 à 3. On se placera dans un repère allant de -2 à 4 pour les abscisses et de -8 à 8 pour les ordonnées. On fera apparaitre aussi une légende en haut à gauche.