Parcours 6. Pour aller toujours plus loin…

Rappelons qu'il est possible de représenter une matrice par une « liste de listes », correspondant à une représentation ligne par ligne. Ainsi, si **M** *est une telle liste de listes, l'élément désigné par* **M[2][5]** *correspond à l'élément* **M[2, 5]** *de la matrice* **M** *(voir Parcours 3).*

Exercice 47. (Transposition de matrice.) Écrire une fonction **transposerMatrice** qui prend comme argument une liste non vide de listes représentant une matrice et renvoie une liste de listes représentant la transposée de cette matrice.

Exercice 48. (Déterminant d'une matrice.) Écrire une fonction **déterminantMatrice** qui prend comme argument une liste non vide de listes représentant une matrice carrée et renvoie le déterminant de cette matrice. (On n'hésitera pas à utiliser ici une fonction *récursive*, c'est-à-dire qui fait appel à elle-même, mais sur une matrice de dimension moindre…).

Exercice 49. (Comatrice.) Écrire une fonction **comatrice** qui prend comme argument une liste non vide de listes représentant une matrice carrée et renvoie la comatrice de cette matrice.

Exercice 50. (Matrice inverse.) Écrire une fonction **matriceInverse** qui prend comme argument une liste non vide de listes représentant une matrice carrée et renvoie la matrice inverse de cette matrice.

Exercice 51. (Points alignés.) Si on note $(X_A ; Y_A)$, $(X_B ; Y_B)$ et $(X_C ; Y_C)$ les coordonnées de trois points A, B et C, et deter(A, B, C) la valeur définie par deter(A, B, C) = $(X_B – X_A)(Y_C – Y_A) – (Y_B – Y_A)(X_C – X_A)$, alors ces trois points sont alignés si et seulement si cette valeur est nulle.

Écrire une fonction **deter** qui prend comme arguments trois listes représentant les coordonnées de trois points A, B et C, et renvoie la valeur de deter(A, B, C). (Cette fonction va être également utile pour les exercices suivants.)

Écrire une fonction **pointsAlignes** qui prend comme arguments trois listes représentant les coordonnées de trois points A, B et C, et détermine si ces trois points sont ou non alignés.

Exercice 52. (Points en position générale.) On dit que n points P_1, …, P_n sont en position générale si on ne peut trouver parmi eux trois points alignés. Écrire une fonction **enPositionGenerale** qui prend comme argument une liste de listes représentant les coordonnées d'en ensemble de points et détermine si ces points sont ou non en position générale.

Exercice 53. (Polygone convexe.) On souhaite écrire une fonction qui détermine si un polygone est convexe ou non. Le polygone est représenté sous forme d'une liste de listes, correspondant à la liste des coordonnées des points P_1, …, P_n ($n \geq 2$) formant le polygone. (Par exemple, la liste **[[1, 3], [1, 5], [4, 5], [4, 3]]** définit les sommets d'un rectangle.)

Rappelons qu'un tel polygone est convexe si et seulement si, pour toute paire de points consécutifs (P_i, P_{i+1}) (les points P_n et P_1 sont considérés comme étant consécutifs), tous les autres points sont du même côté de la droite déterminée par ces deux points (ou sont sur cette même droite). Cela revient à vérifier que pour chacun des points P_k, $k \neq i$, $k \neq i+1$, la valeur deter(P_i, P_{i+1}, P_k) définie précédemment est toujours de même signe… ou nulle.

Écrire une fonction **deter** qui prend comme arguments trois listes représentant les coordonnées de trois points A, B et C, et renvoie la valeur de deter(A, B, C).

Écrire une fonction **estConvexe** qui prend comme argument une liste de listes représentant un polygone et détermine si ce polygone est ou non convexe.

Exercice 54. (Interpolation de Lagrange.) On se donne n + 1 points P_0, …, P_n, de coordonnées (x_0,y_0), …, (x_n,y_n) tels que les X_i sont distincts deux à deux. Les polynômes de Lagrange associés à ces points sont définis ainsi :

$$l_i(X) = \prod_{j=0, j\neq i}^{n} \frac{X - x_j}{x_i - x_j} = \frac{X - x_0}{x_i - x_0} \cdots \frac{X - x_{i-1}}{x_i - x_{i-1}} \frac{X - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{X - x_n}{x_i - x_n}.$$

On veut construire le polynôme de degré minimal dont la courbe passe par tous ces points. La solution est donnée par le polynôme d'interpolation, L(X), défini ainsi :

$$L(X) = \sum_{j=0}^{n} y_j \left(\prod_{i=0, i\neq j}^{n} \frac{X - x_i}{x_j - x_i} \right)$$

Ce polynôme est l'unique polynôme de degré n vérifiant $L(x_i) = y_i$ pour tout i.

Écrire une fonction **polynomeLagrange** qui prend comme arguments une liste d'au moins trois listes, représentant les coordonnées d'au moins trois points, un entier **i** et un flottant **x**, et renvoie la valeur de $\mathbf{l_i(x)}$.

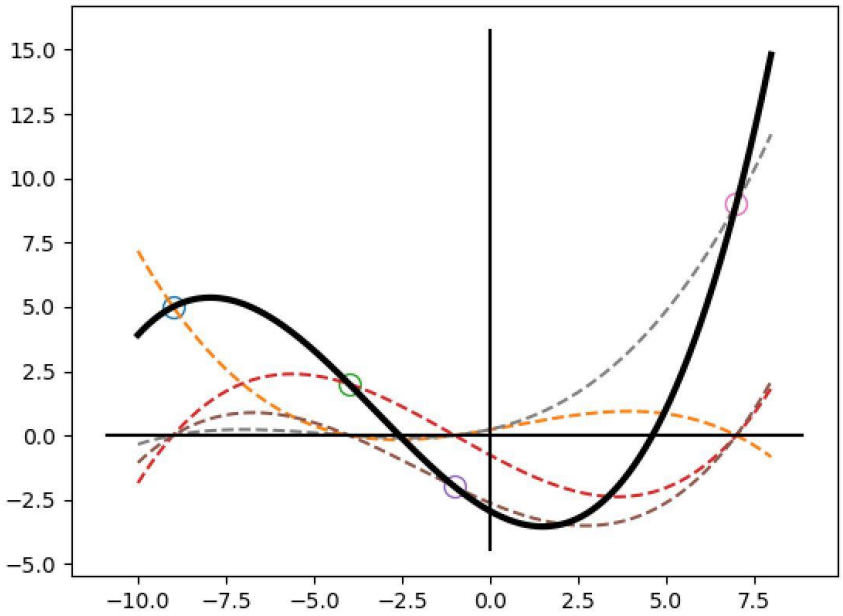
Écrire une fonction **dessineLagrange** qui prend comme arguments une liste d'au moins trois listes, représentant les coordonnées d'au moins trois points, un entier **i**, deux flottants **a** et **b**, et dessine la courbe de la fonction $\mathbf{x \to l_i(x)}$ sur l'intervalle **[a ; b]**.

Écrire une fonction **polynomeInterpolation** qui prend comme arguments une liste d'au moins trois listes, représentant les coordonnées d'au moins trois points, un flottant **x**, et renvoie la valeur de **L(x)**.

Écrire une fonction **dessineLagrange** qui prend comme arguments une liste d'au moins trois listes, représentant les coordonnées d'au moins trois points, deux flottants **a** et **b**, et dessine la courbe de la fonction $\mathbf{x \to L(x)}$ sur l'intervalle **[a ; b]**.

Enfin, écrire une fonction **representationLagrange** qui prend comme arguments une liste d'au moins trois listes, représentant les coordonnées d'au moins trois points, dessine les courbes des polynômes de Lagrange associées à ces points ainsi que la courbe de la fonction $\mathbf{x \to L(x)}$. Toutes ces courbes seront dessinées sur l'intervalle **[mini ; maxi]**, où **mini** correspond à l'abscisse minimale des points $\mathbf{P_i}$ moins un, et **maxi** à l'abscisse maximale de ces points plus un.

Exemple. Appelée avec la liste **[[-9,5], [-4,2], [-1,-2], [7,9]]**, la fonction **representationLagrange** produira le graphique ci-dessous.



Parcours 7. Gestion de stock

Une entreprise fabrique des lampes à partir des pièces suivantes : des ampoules LED, des ampoules halogènes, des douilles simples, des douilles multiples, des interrupteurs simples et des (interrupteurs) gradateurs.

Elle peut fabriquer trois types de lampe dont les protocoles de fabrication sont les suivants :

- lampe L1 : deux ampoules LED, deux douilles simples, un interrupteur simple ;
- lampe L2 : quatre ampoules halogènes, deux douilles multiples, deux gradateurs ;
- lampe L3 : trois ampoules LED, deux ampoules halogènes, deux douilles multiples, un interrupteur simple.

Pour chaque pièce, on dispose d'une certaine quantité en stock. L'entreprise reçoit des commandes de ses clients, chaque commande précisant la quantité de lampes à réaliser pour chacun des types de lampe proposés. L'entreprise reçoit également des livraisons de ses fournisseurs (quantité de pièces pour chaque pièce).

On souhaite disposer de fonctions permettant :

- de mettre à jour le stock à partir d'une livraison reçue,
- de savoir combien de lampes d'un type donné il est possible de fabriquer en fonction du stock de pièces disponible,
- de savoir si une commande est réalisable, en fonction du stock de pièces disponible,
- de connaître le réapprovisionnement nécessaire (quantité de pièces à commander) pour réaliser une commande actuellement non réalisable,
- de mettre à jour le stock à partir d'une commande réalisable (les pièces nécessaires à la fabrication sont retirées du stock),
- rajouter un nouveau type de lampe au catalogue, ainsi que le protocole de fabrication associé,

Le travail demandé est le suivant :

1. Comment représenter, à l'aide de listes ou de listes de listes, les éléments suivants : le catalogue (les types de lampe proposés), les protocoles de fabrication, les pièces, le stock, les commandes, les livraisons ? (Certains de ces éléments peuvent être représentés au sein d'une même liste.)
2. Écrire les fonctions nécessaires à cette gestion de stock.

Parcours 8. Traitement d'images (équipe pédagogique informatique semestre 1, univ. de Bordeaux)

Une image, en informatique, est un simple tableau à deux dimensions de points colorés (appelés pixels, pour *picture elements*). Les coordonnées (x, y) d'un pixel expriment sa position au sein de l'image : x est son abscisse, en partant de la gauche de l'image, et y est son ordonnée, en partant du haut de l'image (à l'inverse de l'ordonnée mathématique, donc). Elles partent toutes deux de 0. Le schéma ci-dessous montre un exemple d'image en gris sur fond blanc, de 7 pixels de largeur et 4 pixels de hauteur, les abscisses vont donc de 0 à 6 et les ordonnées de 0 à 3.

0,0	1,0	2,0	3,0	4,0	5,0	6,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3

La couleur **RGB(r, g, b)** d'un pixel est la quantité de rouge (**r** pour *red*), vert (**g** pour *green*) et de bleu (**b** pour *blue*) composant la couleur affichée à l'écran. Le mélange se fait comme trois lampes colorées rouge, verte et bleue, et les trois valeurs **r**, **g**, **b** expriment les intensités lumineuses de chacune de ces trois lampes, exprimées entre 0 et 255. Par exemple, **(0, 0, 0)** correspond à trois lampes éteintes, et produit donc du noir, **(255, 255, 255)** correspond à trois lampes allumées au maximum, et produit donc du blanc, **(255, 0, 0)** correspond à seulement une lampe rouge allumée au maximum, et produit donc du rouge, **(255, 255, 0)** correspond à une lampe rouge et une lampe verte allumées au maximum, et produit donc du jaune. Et ainsi de suite. Cela correspond très précisément à ce qui se passe sur vos écrans : si vous prenez une loupe, vous verrez que l'écran est composé de petits points (appelés sous-pixels) verts, rouges et bleus, allumés plus ou moins fortement pour former les couleurs.

Partie 1 : Mise en jambe

Python permet bien sûr de manipuler des images, à l'aide de la *Python Imaging Library* (**PIL**), qui est préinstallée sur vos machines. Il faudra donc, pour ce parcours, importer ce module : **import PIL.Image as PIL**. On dispose alors d'un ensemble de fonctions permettant de manipuler des images. Voici un résumé des fonctions que nous utiliserons :

PIL.open(nomfichier)	Ouvre le fichier « nomfichier » et retourne l'image qu'il contient (par exemple open("teapot.png")).
PIL.new("RGB", (largeur, hauteur))	Retourne une image de taille largeur × hauteur, initialement noire.
PIL.Image.show(img)	Affiche l'image img.
PIL.Image.putpixel(img, (x, y), (r, g, b))	Peint le pixel (x, y) dans l'image img de la couleur (r, g, b)

1. Exécuter les instructions suivantes (penser à importer d'abord le module **PIL.Image**) :

```
monImage = PIL.new ("RGB", (300, 200))
PIL.Image.putpixel (monImage, (50, 50), (255, 255, 255))
PIL.Image.show (monImage)
```

- Observer ce qu'effectue chaque instruction, et l'image produite. Confirmer ainsi le sens dans lequel fonctionnent les coordonnées.
- Peindre le pixel de coordonnées (250, 150) en vert (ne pas oublier d'appeler à nouveau **PIL.Image.show (monImage)** pour afficher le résultat).
- Peindre le pixel de coordonnées (50, 150) en violet.
- Exécuter l'instruction suivante et observer le contenu des variables largeur et hauteur.

```
largeur, hauteur = monImage.size
```

- Récupérer l'image de la théière de l'Utah et la sauvegarder dans le dossier des fichiers python. Exécuter les instructions suivantes :

```
theiere = PIL.open ("teapot.png")
PIL.Image.show (theiere)
```

Vous pourrez facilement travailler sur cette photo. Vous pourrez bien sûr, pour tous les exercices, utiliser d'autres images venant d'Internet ou d'ailleurs.

2. Écrire une fonction **ligneHorizontaleBlancheAuMilieu(img)** (utilisant une boucle for) qui dessine une ligne horizontale blanche au milieu de l'image **img** (séparant donc l'image donnée en deux parties). La tester sur la théière, sur une image noire (générée par un appel à **PIL.new**), ainsi que sur au moins une autre image.

3. Écrire une fonction **ligneHorizontaleAuMilieu(img, c)** qui permet cette fois choisir la couleur de la ligne lors de l'appel. L'argument **c** est un triplet de valeurs, de la forme (r, g, b).
On pourra donc tester cette fonction en écrivant, par exemple, **ligneHorizontaleAuMilieu(img, (255, 255, 0))**.

4. Écrire une fonction **ligneHorizontale(img, c, y)** qui dessine, dans une image **img** donnée, une ligne horizontale de couleur **c** à la distance **y** du haut de l'image.
Donner une nouvelle version de la fonction **ligneHorizontaleAuMilieu2(img, c)** qui dessine une ligne horizontale de couleur **c** au milieu de l'image **img** en ne faisant qu'appeler la fonction **ligneHorizontale(img, c, y)**.

5. En faisant appel à la fonction **ligneHorizontale(img, c, y)**, écrire une fonction **grilleHorizontale(img, c, d)** qui dessine une grille de lignes horizontales espacées de **d** pixels.
Par exemple, **grilleHorizontale(monImage, (255, 255, 0), 10)** dessinerait dans **monImage** des lignes jaunes ayant pour ordonnées 0, 10, 20, 30, etc.

   Ne pas hésiter à profiter de la fonction **range(debut, fin, pas)** qui est toute prête à fournir la liste exacte des ordonnées concernées…

6. Écrire une fonction **diagonale(img)** qui dessine en blanc la ligne diagonale entre le coin en haut à gauche et le coin en bas à droite de l'image. Tester avec différentes tailles d'image, notamment avec des images dont la hauteur est supérieure à la largeur…

7. Écrire une fonction **cercle(img, x, y, r)** qui dessine un cercle de rayon **r** dont le centre a pour coordonnées **(x ; y)**.
Rappelons que les fonctions trigonométriques (**math.cos, math.sin, math.pi**…) sont disponibles en important le module **math** (**import math**).

Partie 2 : Manipulation de couleurs

Jusqu'ici on a seulement peint des pixels, mais on peut également récupérer la couleur d'un pixel :

PIL.Image.getpixel (img, (x, y))	Retourne la couleur du pixel (x, y) dans l'image img

Il est courant d'appliquer un filtre sur une image, c'est-à-dire un calcul sur chacun des pixels de l'image. L'instruction :

(r, g, b) = PIL.Image.getpixel (img, (10, 10))

récupère les valeurs **r**, **g** et **b** de la couleur du pixel de coordonnées **(10, 10)**. Par exemple, pour ne conserver que la composante rouge du pixel **(10, 10)**, on écrira :

PIL.Image.putpixel (img, (10, 10), (r, 0, 0))

8. Écrire une fonction **filtreRouge(img)** qui, pour chaque pixel de l'image, ne conserve que la composante rouge. La tester sur la photo de théière.
Faire de même pour le vert et le bleu et afficher les trois résultats ainsi que l'image d'origine côte à côte. Remarquer notamment que pour le bleu il n'y a pas d'ombre en bas à droite. En effet, la source lumineuse en haut à gauche ne contient pas de bleu…

   Pour recopier une image **img** dans une image plus grande, par exemple **imgGlobal**, en position **(x,y)** (correspondant aux coordonnées du coin supérieur gauche) on écrira :

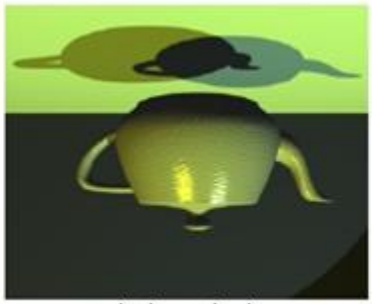
imgGlobal.paste(img, [x, y])

9. Écrire une fonction **modifierLuminosite(img, facteur)** qui, pour chaque pixel, multiplie par le flottant **facteur** les valeurs des trois composantes r, g, et b. Remarquer que la fonction **PIL.Image.putpixel** n'apprécie pas que l'on donne des valeurs non entières. Utiliser donc la fonction **math.floor(f)** qui convertit une valeur flottante **f** en une valeur entière (arrondi par défaut).
   Tester les facteurs 1.2, 2, 0.8, 0.5 sur la photo de théière (ne pas oublier de recharger la photo à chaque fois pour éviter de cumuler les effets).
10. Écrire une fonction **monochrome(img)** qui, pour chaque pixel, calcule la moyenne **m = (r + b + g) // 3** des composantes r, g, b, et peint le pixel de la couleur **(m, m, m)**.
    En observant bien, les éléments verts semblent cependant plus foncés que sur la photo d'origine. L'œil humain est effectivement peu sensible au bleu et beaucoup plus au vert. Une conversion plus « ressemblante » est obtenue en utilisant plutôt **m = 0.3 ∗ r + 0.59 ∗ g + 0.11 ∗ b**. Essayer et constater que les éléments verts ont une luminosité plus fidèle à la version couleur.
11. Écrire une fonction **noirEtBlanc(img)** qui convertit une image monochrome (telle que produite par la fonction monochrome de la question 10) en une image noir et blanc : chaque pixel peut valoir (0, 0, 0) ou (255, 255, 255) selon que la luminosité (**r + g + b**) est plus petite ou plus grande que **3 * 127**.
    Cette conversion ne permet néanmoins pas de représenter les variations douces de luminosité.
12. L'effet de *posterisation* est obtenu en diminuant le nombre de couleurs d'une image. Une manière simple de l'obtenir est d'arrondir les composantes r, g, b des pixels de l'image à des multiples d'un entier, par exemple des multiples de 64.
    Écrire une fonction **posteriser(img, n)** qui arrondit les composantes des pixels de l'image à un multiple de **n**. Essayer sur une photo avec **n = 64, 128, 150, 200**…
13. Une manière simple de rendre une image floue est d'utiliser l'opération moyenne. Écrire une fonction **flou(img)** qui, pour chaque pixel n'étant pas sur les bords, le peint de la couleur moyenne des pixels voisins.
    Comparer le résultat à l'original. Pourquoi faudrait-il plutôt construire une deuxième image, que renverra la fonction, et ne pas toucher à la première image ?

Partie 3 : Manipulations géométriques

14. L'objectif de cet exercice est d'écrire le code de deux fonctions Python permettant d'effectuer la transformation miroir (symétrie axiale) d'une image. En voici un exemple :


image initiale


miroir vertical


miroir horizontal

    Écrire une fonction **miroirVertical(img)** qui renvoie l'image correspondant au miroir vertical de **img**.

    Écrire une fonction **miroirHorizontal(img)** qui renvoie l'image correspondant au miroir horizontal de **img**.
15. Écrire une fonction **rotation90(img)** qui renvoie une copie de l'image **img** tournée de 90 degrés vers la droite, c'est-à-dire dans le sens horaire.
    Pourquoi est-on obligé d'utiliser une deuxième image, plutôt que faire la rotation « en place », c'est-à-dire en n'utilisant qu'une image ?
16. Écrire une fonction **rotationImage(img, angle)** qui renvoie une copie de l'image **img** tournée de **angle** radians vers la droite.