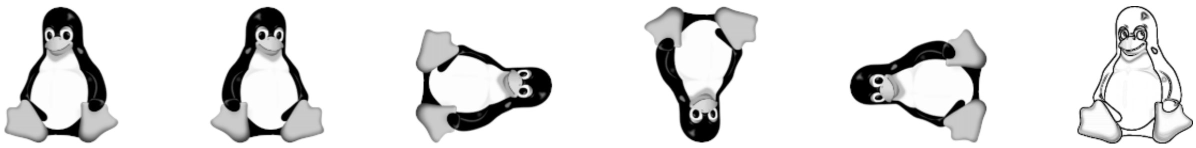
TP 9 d'informatique - MPSI - 16/12/2025 - Manipulation d'images



Le but de ce TP est d'utiliser Python pour manipuler des images au format `jpg` ou `png`. On s'intéresse ici à des "images matricielles" (par opposition aux "images vectorielles" du format `pdf`), c'est à dire qu'une image est composée de "pixels" répartis sur une grille de `L` colonnes et `H` lignes.

Dans ce sujet, les couleurs des pixels sont décrites à l'aide du codage RGB (Red Green Blue). Une couleur est donc un triplet d'entiers (r, g, b) tel que $0 \leqslant r, g, b \leqslant 255$, où r correspond à la quantité de rouge, g à la quantité de vert et b à la quantité de bleu. Par exemple, la couleur $(0, 0, 255)$ est un bleu pur, $(255, 255, 0)$ est un jaune (mélange de rouge et vert), $(0, 0, 0)$ est le noir et $(255, 255, 255)$ le blanc.

Soient $(H, L) \in (\mathbb{N}^*)^2$. On dit qu'une image est de dimensions $H \times L$ si les pixels forment une grille de H lignes et L colonnes. En Python, l'image sera représentée par une liste de listes de triplets d'entiers « `mat: list[list[int,int,int]]` » où pour tout $\texttt{i} \in [\![\texttt{0}, \texttt{H}-\texttt{1}]\!]$, et tout $\texttt{j} \in [\![\texttt{0}, \texttt{L}-\texttt{1}]\!]$, le triplet `mat[i][j]` est le codage RGB du pixel de coordonnées `(i,j)`. Ici, l'entier `i` est l'indice de la ligne (`i = 0` correspond à la ligne du haut et `i = H-1` à la ligne du bas) et l'entier `j` est l'indice de la colonne (`j = 0` correspond à la colonne de gauche et `j = L-1` à la colonne de droite). Voici un exemple d'image de dimensions 2×3 :

```
mat = [
    [(255,0,0),(0,255,0),(0,0,255)],
    [(255,255,0),(0,0,0),(255,255,255)]
]
```

Rouge	Vert	Bleu
Jaune	Noir	Blanc

Récupérez l'archive `zip` sur la page web du cours et décompressez la dans le dossier de votre choix (si vous êtes sur un ordinateur du lycée, utilisez comme d'habitude votre répertoire personnel `U:/`) :

https://informatique-lhp.fr/itc-mpsi.html

Vérifiez que le dossier `TP09_images` contient bien cinq images, puis exécutez le fichier `TP09.py` et observez le résultat. Le fichier source `TP09.py` contient plusieurs fonctions permettant de manipuler des images à l'aide du module `PIL.Image` :

→ La fonction de signature « `get_mat(file_in: str) -> list[list[int,int,int]]` » qui prend en entrée le chemin vers un fichier `png` ou `jpg` et renvoie la matrice RGB correspondante.

→ La fonction de signature « `save(mat: list[list[int,int,int]], file_out: str) -> NoneType` » qui enregistre l'image associée à une matrice RGB dans le fichier dont le chemin est `file_out`.

→ La fonction de signature « `copy(file_in: str, file_out: str) -> NoneType` » qui copie l'image contenue dans le fichier `file_in` vers le fichier `file_out`. Notez qu'il est possible de modifier l'extension du fichier copié (`png` vers `jpg` ou inversement). Cette fonction ne sera pas utile dans la suite du TP, elle sert à illustrer l'utilisation des deux fonctions précédentes.

Dans tout le TP, vous devez utiliser les fonctions `get_mat` et `save`, mais pas directement le module `PIL.Image`. Même lorsque ce n'est pas précisé, les fonctions que vous allez écrire prennent en entrée les chaînes de caractères `file_in` et `file_out`. Le fichier `file_in` contient l'image à modifier et le résultat devra être enregistré dans le fichier `file_out`. On pourra supposer que toutes les images contiennent au moins une ligne et une colonne de pixels.

L'image `est_premier.jpg` sera utilisée uniquement dans l'exercice 4 et `pierres.png` dans l'exercice 6.

Exercice 1. Modification d'une image pixel par pixel

Étant donnée une couleur codée par un triplet `(r,g,b)`, la ***couleur inverse*** est codée par le triplet `(255-r, 255-g, 255-b)`. Le négatif d'une image est l'image dans laquelle chaque couleur est remplacée par son inverse. Voici un exemple d'image de dimensions 3×2 et son négatif :

```
[[(175, 61 , 203), (195, 93 , 208)],
 [(42 , 221, 67 ), (80 , 196, 190)],
 [(43 , 89 , 56 ), (13 , 218, 211)]]
```

```
[[(80 , 194, 52 ), (60 , 162, 47 )],
 [(213, 34 , 188), (175, 59 , 65 )],
 [(212, 166, 199), (242, 37 , 44 )]]
```

1. Écrire une fonction qui créé le négatif d'une image. Vérifiez que vous obtenez la même image lorsque vous appliquez deux fois la fonction.

De manière plus générale, on peut appliquer sur chaque pixel une fonction

`f(r: int, g: int, b: int) -> (int, int, int).`

2. Écrire une fonction `filtre` qui prend en entrée une fonction `f` et applique `f` sur chacun des pixels de l'image. Tester avec une fonction `f` permettant d'obtenir le négatif d'une image ainsi qu'avec la fonction `f` : `(r,g,b)` $\mapsto$ `(g,b,r)`.

Exercice 2. Réduction et agrandissement

Pour réduire une image d'un facteur $d \in \mathbb{N}^*$, la manière la plus simple est de ne conserver que les pixels de coordonnées `(i,j)` tels que `i` et `j` sont divisibles par d. Pour $d = 2$, voici un exemple d'image de dimensions 4×5 et sa version réduite de dimensions 2×3 :

```
[[(47 , 92 , 246), (207, 61 , 202), (178, 145, 17 ), (140, 54 , 124), (211, 109, 39 )],
 [(113, 118, 244), (141, 46 , 166), (115, 16 , 5  ), (62 , 209, 234), (211, 93 , 174)],
 [(6  , 224, 160), (47 , 177, 6  ), (18 , 213, 27 ), (67 , 147, 150), (252, 112, 245)],
 [(219, 246, 118), (121, 244, 178), (9  , 67 , 86 ), (78 , 97 , 24 ), (2  , 165, 161)]]
```

```
[[(47 , 92 , 246), (178, 145, 17 ), (211, 109, 39 )],
 [(6  , 224, 160), (18 , 213, 27 ), (252, 112, 245)]]
```

1. (a) Si l'image initiale est de dimensions `H` $\times$ `L`, quelle sont les dimensions de l'image finale ? (Attention : une simple division entière par d n'est pas suffisant)
   (b) Écrire une fonction qui réduit l'image donnée en entrée d'un facteur d.

Pour agrandir une image d'un facteur $d \in \mathbb{N}^*$, on copie chaque ligne et chaque colonne d fois. Pour $d = 2$, voici un exemple d'image de dimensions 3×2 et sa version agrandie de dimensions 6×4 :

```
[[(111, 189, 82 ), (70 , 183, 53 )],
 [(119, 64 , 52 ), (51 , 53 , 225)],
 [(31 , 109, 16 ), (248, 83 , 64 )]]
```

```
[[(111, 189, 82 ), (111, 189, 82 ), (70 , 183, 53 ), (70 , 183, 53 )],
 [(111, 189, 82 ), (111, 189, 82 ), (70 , 183, 53 ), (70 , 183, 53 )],
 [(119, 64 , 52 ), (119, 64 , 52 ), (51 , 53 , 225), (51 , 53 , 225)],
 [(119, 64 , 52 ), (119, 64 , 52 ), (51 , 53 , 225), (51 , 53 , 225)],
 [(31 , 109, 16 ), (31 , 109, 16 ), (248, 83 , 64 ), (248, 83 , 64 )],
 [(31 , 109, 16 ), (31 , 109, 16 ), (248, 83 , 64 ), (248, 83 , 64 )]]
```

2. Écrire une fonction qui agrandit l'image donnée en entrée d'un facteur d.

Exercice 3. Symétrie et rotation

Une symétrie axiale verticale consiste à appliquer une symétrie par rapport à la droite verticale qui passe au milieu de l'image. Par exemple à partir du 1$^{\text{er}}$ pingouin du début de l'énoncé, on obtient le 2$^{\text{ème}}$ pingouin.

1. Écrire une fonction qui effectue une symétrie axiale verticale sur l'image donnée en entrée. Si besoin, on pourra lire les indications ci-dessous.

On souhaite maintenant faire une rotation d'un quart de tour dans le sens des aiguilles d'une montre. Par exemple à partir du 1$^{\text{er}}$ pingouin du début de l'énoncé, on obtient le 3$^{\text{ème}}$ pingouin.

2. Écrire une fonction qui effectue la rotation sur l'image donnée en entrée. Vérifiez que le pingouin obtenu regarde vers le haut. Si besoin, on pourra lire les indications ci-dessous.

Indications (essayez de faire l'exercice sans lire ce qui suit). On répondra d'abord aux questions intermédiaires ci-dessous dans le cas de la symétrie puis de la rotation.

(a) Supposons que l'image initiale soit de dimensions `H` × `L`. Quelles sont les dimensions de l'image finale ?

(b) Pour chacun des pixels se trouvant dans l'un des quatre coins de l'image initiale, donner les coordonnées du pixel avant et après la transformation.

(c) Soient `(i,j)` les coordonnées d'un pixel quelconque de l'image initiale. Donner les coordonnées de ce pixel dans l'image finale.

Exercice 4. Image en niveaux de gris

Dans le codage RGB, les différentes teintes de gris correspondent à des valeurs de la forme (r, g, b) où $r = g = b$. Une première méthode pour convertir une image en niveaux de gris, consiste à fixer trois flottants `c1, c2, c3` tels que `c1 + c2 + c3 = 1`, puis à remplacer chaque pixel de couleur `(r,g,b)` par `(m,m,m)` où `m = c1 × r + c2 × g + c3 × b`.

1. Écrire une fonction qui prend en entrée `c1, c2, c3` et convertit une image en niveaux de gris en utilisant la méthode décrite ci-dessus. On testera dans les cas suivants :
   - → Les trois composantes RGB ont la même pondération : `c1 = c2 = c3 = 1/3`.
   - → On tient compte uniquement de la composante verte : `c1 = c3 = 0` et `c2 = 1`.
   - → On utilise la recommandation 709 : `c1 = 0.2126`, `c2 = 0.7152` et `c3 = 0.0722`.

Une autre approche consiste à fixer `m` comme étant la moyenne de la composante la plus forte et de la composante la plus faible.

2. Écrire une fonction qui convertit une image en niveaux de gris en utilisant cette deuxième méthode.

Pour obtenir une image en noir et blanc, on fixe un entier noté `seuil` et on convertit l'image en niveaux de gris (en général avec la recommandation 709). Les pixels dont le paramètre `m` vérifie `m` ⩽ `seuil` sont coloriés en noir et les autres pixels sont coloriés en blanc.

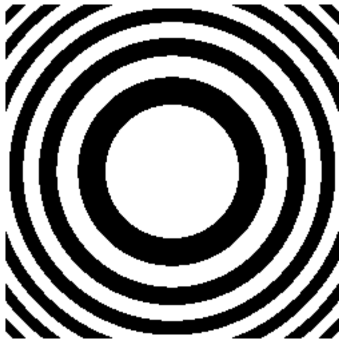
3. (a) Écrire une fonction qui prend en entrée une image en niveaux de gris (c'est à dire dont les composantes RGB sont de la forme `(m,m,m)`) et la convertit en noir et blanc.
   (b) L'image `est_premier.jpg` est la photo d'un programme Python. Déterminer un seuil permettant d'améliorer la lisibilité de cette image.

Exercice 5. Aliasing

L'aliasing ou ***repliement de spectre*** se produit lorsque la résolution d'une image est trop faible par rapport aux motifs qui s'y trouvent. Ce phénomène induit des aberrations comme dans l'image ci-contre qui provient du site :

https://vincmazet.github.io/signal1/numerisation/echantillonnage.html

Le but de cet exercice est d'illustrer ce phénomène. Soit $R \in \mathbb{R}_+^*$. Nous allons colorier les pixels d'une image carrée en noir et blanc de manière à obtenir une série d'anneaux concentriques. Pour chaque pixel, on calcule d sa distance avec le centre de l'image, puis :



→ Si $d < R$, le pixel est colorié en blanc.

→ Si $R \leqslant d < R\sqrt{2}$, le pixel est colorié en noir.

→ Si $R\sqrt{2} \leqslant d < R\sqrt{3}$, le pixel est colorié en blanc.

→ Si $R\sqrt{3} \leqslant d < R\sqrt{4}$, le pixel est colorié en noir ...

De manière générale, si $R\sqrt{n} \leqslant d < R\sqrt{n+1}$ avec n un entier pair alors le pixel est colorié en blanc, sinon il est colorié en noir.

1. Expliquer comment déterminer en temps constant la couleur d'un pixel.
2. En déduire une fonction qui prend entrée R ainsi que la taille de l'image et crée l'image décrite ci-dessus. Lorsque $R = 50$, vous devez observer les anneaux concentriques. En revanche, lorsque R devient petit, la résolution de l'image n'est plus suffisante pour voir les anneaux qui sont remplacés par d'autres motifs. On testera notamment avec $0 < R < 1$.

Exercice 6.

À l'aide de Python, compter le nombre de pierres sur l'image `pierres.png` (il y en a 46).

Exercice 7. Transformation du photomaton

Dans cet exercice, on ne considère que des images dont la hauteur `H` et la largeur `L` sont paires. La transformation du photomaton se définit comme suit :

- On découpe l'image en carrés de dimensions 2×2.
- On construit quatre nouvelles images de hauteur `H//2` et de largeur `L//2` :
  - L'image en haut à gauche est constituée des pixels qui se trouvent en haut à gauche des carrés de dimensions 2×2.
  - Idem en remplaçant les deux "en haut à gauche" par "en haut à droite", puis "en bas à gauche" et enfin "en bas à droite".

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)

Image initiale

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)

Découpage en carrés de dimension 2×2

(0,0)	(0,2)	(0,4)	(0,6)	(0,1)	(0,3)	(0,5)	(0,7)
(2,0)	(2,2)	(2,4)	(2,6)	(2,1)	(2,3)	(2,5)	(2,7)
(4,0)	(4,2)	(4,4)	(4,6)	(4,1)	(4,3)	(4,5)	(4,7)
(1,0)	(1,2)	(1,4)	(1,6)	(1,1)	(1,3)	(1,5)	(1,7)
(3,0)	(3,2)	(3,4)	(3,6)	(3,1)	(3,3)	(3,5)	(3,7)
(5,0)	(5,2)	(5,4)	(5,6)	(5,1)	(5,3)	(5,5)	(5,7)

Image finale

1. Écrire une fonction qui prend en entrée un entier $n \in \mathbb{N}$ et applique la transformation du photomaton n fois à une image.

Si on applique la transformation du photomaton suffisamment de fois, on fini par retrouver l'image initiale. On appelle ***période de retour*** le nombre minimal d'itérations nécessaires pour retrouver l'image initiale.

2. Écrire une fonction qui renvoie la période de retour de l'image donnée en entrée. Pour que les tests s'exécutent en un temps raisonnable, on pourra utiliser des images carrées. Par exemple, la période de retour d'une image de dimensions 400×400 est 18.

Exercice 8. Rognage

"Rogner une image" consiste à ne conserver qu'une partie de l'image. Écrire une fonction de signature :

```
rogner(file_in: str, file_out: str, i0: int, j0: int, H0: int, L0: int) -> NoneType
```

qui créé une nouvelle image telle que :

→ Le pixel en haut à gauche est celui de coordonnées `(i0,j0)` dans l'image initiale.

→ Cette nouvelle image est de dimensions `H0` × `L0`.

Par exemple, lorsqu'on rogne l'image `pinguoin.png` avec les paramètres `i0 = 178`, `j0 = 140`, `H0 = 77` et `L0 = 74`, on obtient la patte palmée gauche du pingouin. On vérifiera que les valeurs de `(i0, j0, H0, L0)` sont cohérentes avec la taille de l'image à l'aide d'un ou plusieurs `assert`.

Exercice 9. Convolution et détection de contours

Convolution. Soient M et N deux matrices (c'est à dire des listes de listes de flottants). Plus tard, la matrice M représentera l'une des trois composantes RGB d'une image. La matrice N s'appelle le ***noyau*** ; elle contient n sous-listes de taille n où $n \in \mathbb{N}$ est un entier impair. Le produit de convolution de M et N est une matrice notée R ayant les mêmes dimensions que M. Pour calculer chaque $R[i][j]$, on imagine superposer M et N en plaçant l'élément central de N sur $M[i][j]$ (cet élément central existe car n est impair). La valeur de $R[i][j]$ est alors la somme de tous les produits ab où a est un élément de N et b est l'élément de M qui lui est superposé. Pour gérer le cas où l'un des a n'est superposé à aucun élément, on duplique les valeurs en bordure de M. Par exemple, avec :

$$N = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \text{et} \qquad M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{bmatrix}$$

on obtient M' la matrice dans laquelle les valeurs en bordure de M ont été dupliquées et R le résultat du produit de convolution :

$$M' = \begin{bmatrix} 1 & 1 & 2 & 3 & 4 & 5 & 5 \\ 1 & 1 & 2 & 3 & 4 & 5 & 5 \\ 6 & 6 & 7 & 8 & 9 & 10 & 10 \\ 11 & 11 & 12 & 13 & 14 & 15 & 15 \\ 16 & 16 & 17 & 18 & 19 & 20 & 20 \\ 16 & 16 & 17 & 18 & 19 & 20 & 20 \end{bmatrix} \qquad \text{et} \qquad R = \begin{bmatrix} -5 & -3 & -2 & -1 & 1 \\ 5 & 7 & 8 & 9 & 11 \\ 10 & 12 & 13 & 14 & 16 \\ 20 & 22 & 23 & 24 & 26 \end{bmatrix}$$

Les coefficients 13, -5 et -2 de R sont obtenus par les calculs :

$$\begin{array}{ccccccccc} 13 & = & 0\times 7 & + & (-1)\times 8 & + & 0\times 9 & + \\ & & (-1)\times 12 & + & 5\times 13 & + & (-1)\times 14 & + \\ & & 0\times 17 & + & (-1)\times 18 & + & 0\times 19. & \\ \\ -5 & = & 0\times 1 & + & (-1)\times 1 & + & 0\times 2 & + \\ & & (-1)\times 1 & + & 5\times 1 & + & (-1)\times 2 & + \\ & & 0\times 6 & + & (-1)\times 6 & + & 0\times 7. & \\ \\ -2 & = & 0\times 2 & + & (-1)\times 3 & + & 0\times 4 & + \\ & & (-1)\times 2 & + & 5\times 3 & + & (-1)\times 4 & + \\ & & 0\times 7 & + & (-1)\times 8 & + & 0\times 9. & \end{array}$$

1. Écrire une fonction de signature

```
conv(file_in: str, file_out: str, N: list[list[float]]) -> NoneType
```

qui prend en entrée un noyau N et applique le produit de convolution sur chaque composante RGB d'une image. Tester avec les noyaux décrits sur la page wikipédia :

https://fr.wikipedia.org/wiki/Noyau_(traitement_d%27image)

Détection de contours avec la méthode de Sobel. Afin de détecter les contours d'une image, on propose d'utiliser la méthode de Sobel. Dans la suite, la couleur RGB (c, c, c) sera appelée "niveau de gris c" :

→ L'image est d'abord convertie en niveaux de gris à l'aide de la recommandation 709 décrite dans l'exercice 4.

→ On calcule indépendemment deux produits de convolution ayant pour noyaux :

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{et} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

→ Pour chaque pixel de coordonnées (i, j) dans l'image initiale, l'image finale contient un pixel dont le niveau de gris est $255 - \sqrt{a^2 + b^2}$ où a (resp. b) est le niveau de gris du pixel (i, j) dans le résultat de la première convolution (resp. deuxième convolution).

2. Écrire une fonction de signature « `contours(file_in: str, file_out: str) -> NoneType` » qui prend en entrée une image et lui applique la méthode de Sobel. Par exemple à partir du 1$^{\text{er}}$ pingouin du début de l'énoncé, on obtient le 6$^{\text{ème}}$ pingouin.

Exercice 10. Suite de Van Eck

Consignes.

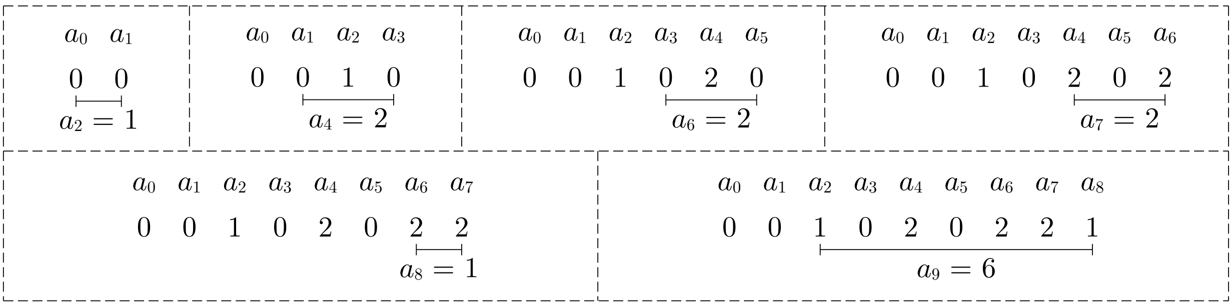
→ Pensez à tester vos fonctions et à laisser les tests dans votre fichier source. Vous perdrez des points si les tests n'apparaissent pas.

→ Des indications sont données à la fin du sujet. Essayez dans un premier temps de résoudre l'exercice sans ces indications.

→ Chaque question comporte également un test facultatif. Si vous souhaitez que votre programme puisse exécuter les tests facultatifs, vous devez concevoir une méthode plus efficace que celle des indications.

Énoncé. La suite de Van Eck $(a_n)_{n\in\mathbb{N}}$ est une suite infinie d'entiers naturels dont les dix premiers termes sont $0, 0, 1, 0, 2, 0, 2, 2, 1, 6$. Cette suite est définie récursivement : $a_0 = 0$ et pour tout $n \geqslant 1$:

- Si a_{n-1} n'apparaît pas dans $L = [a_0, a_1, \ldots, a_{n-2}]$, alors $a_n = 0$.
- Sinon, $a_n = n - m - 1$ où a_m est la dernière apparition de a_{n-1} dans L.



1. Calculer les termes $a_{10}, a_{11}, \ldots, a_{20}$ de la suite de Van Eck.
2. Écrire une fonction `vanEck` qui prend en entrée un entier n et renvoie a_n. Vérifier que $a_{20\,000} = 7$. **Facultatif :** vérifier que $a_{10\,000\,000} = 5\,522\,779$.

Conjectures et théorèmes

Le but de cette partie est de vérifier numériquement plusieurs conjectures et théorèmes concernant la suite de Van Eck.

Théorème 1. *La suite de Van Eck comporte une infinité de* 0.

Pour vérifier le théorème 1, on fixe un entier $k \in \mathbb{N}^*$ et on cherche à déterminer l'indice $z(k)$ tel que $a_{z(k)}$ est le $k^{\text{ème}}$ zéro de la suite de Van Eck. Par exemple :

$$z(1) = 0, \qquad z(2) = 1, \qquad z(3) = 3, \qquad z(4) = 5, \qquad z(5) = 10.$$

3. Écrire une fonction `kemeZero` qui prend en entrée un entier $k \in \mathbb{N}^*$ et renvoie $z(k)$. Votre fonction déclenchera une erreur si $k \leqslant 0$. Vérifier que $z(5000) = 31\,268$. **Facultatif :** vérifier que $z(1\,000\,000) = 7\,930\,437$.

Conjecture 1. *Tous les entiers naturels apparaissent dans la suite de Van Eck.*

Soit $n \in \mathbb{N}$ un entier. On souhaite déterminer le plus petit entier $k_n \in \mathbb{N}$ tel que k_n n'apparaît pas dans l'ensemble $A_n = \{a_0, a_1, a_2, \ldots, a_n\}$. Par exemple, $k_9 = 3$ car c'est le plus petit entier qui n'apparaît pas dans $A_9 = \{0, 0, 1, 0, 2, 0, 2, 2, 1, 6\}$.

4. Écrire une fonction `minAbsents` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie k_n. Vérifier que `minAbsents(20_000)` vaut `379`. **Facultatif :** vérifier que `minAbsents(10_000_000)` vaut `53791`.

Conjecture 2. *Pour tout* $n \in \mathbb{N}^*$*, on pose* $b_n = a_n/n$*. On note également* c_n *le maximum de l'ensemble* $\{b_1, b_2, \ldots, b_n\}$*. Alors* $\lim\limits_{n\to+\infty} c_n = 1$.

5. Écrire une fonction `getC` qui prend en entrée un entier $n \in \mathbb{N}^*$, et renvoie c_n. vérifier que `getC(30_000)` vaut `0.963346...` **Facultatif :** vérifier que `getC(10_000_000)` vaut `0.990816...`

Conjecture 3. *Soit* $A = \Big\{(k_1, k_2) \in \mathbb{N}^2 : (k_1, k_2) \neq (1,1)$ *et* $k_2 \neq k_1 + 1\Big\} \cup \Big\{(0,1)\Big\}$. *Alors pour tout* $(k_1, k_2) \in A$, *il existe* n *tel que* $a_n = k_1$ *et* $a_{n+1} = k_2$.

6. Écrire une fonction de signature « `premiereOccurrence(k1: int, k2: int) -> int` » qui prend en entrée $k_1 \in \mathbb{Z}^2$, $k_2 \in \mathbb{Z}^2$, et renvoie le plus petit entier n tel que $a_n = k_1$ et $a_{n+1} = k_2$. Si $(k_1, k_2) \notin A$, votre fonction déclenchera une erreur. Vérifier que `premiereOccurrence(17, 382)` vaut `27635`. **Facultatif :** vérifier que `premiereOccurrence(31,1502)` vaut `9465773`.

Théorème 2. *Pour tout* $(i,j) \in \mathbb{N}^2$, *si* $i \neq j$, $a_i \neq 0$ *et* $a_j \neq 0$ *alors* $i - a_i \neq j - a_j$.

7. (a) En utilisant uniquement une compréhension de listes, écrire une fonction `listeDiff` qui prend en entrée une liste quelconque $[\mathtt{x_0}, \mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_n}]$, et renvoie une liste contenant tous les $(\mathtt{i} - \mathtt{x_i})$ où $\mathtt{x_i} \neq 0$ et $\mathtt{i} \in [\![\mathtt{0}, \mathtt{n}]\!]$.
   (b) En déduire une fonction `verifTheo2` qui prend en entrée un entier $n \in \mathbb{N}$ et renvoie :
   - `True` si $i - a_i \neq j - a_j$ pour tous les couples $(i,j) \in [\![0, n]\!]^2$ tels que $i \neq j$, $a_i \neq 0$ et $a_j \neq 0$.
   - `False` sinon.

   Dans cette question, vous n'avez pas le droit de considérer que le théorème 2 est connu. Vérifier que `verifTheo2(10_000)` vaut `True`. **Facultatif :** vérifier que `verifTheo2(10_000_000)` vaut `True`.

Indications (essayez d'abord de répondre aux questions sans lire ce qui suit)

2. (a) Écrire une fonction « `termeSuivant(L: list[int]) -> int` » qui prend en entrée une liste de la forme $[\mathtt{a_0}, \mathtt{a_1}, \ldots, \mathtt{a_{n-1}}]$ et renvoie $\mathtt{a_n}$.
   (b) À l'aide de la fonction précédente, écrire une fonction « `listeVanEck(n: int) -> list[int]` » qui renvoie la liste $[\mathtt{a_0}, \mathtt{a_1}, \ldots, \mathtt{a_n}]$.
   (c) En déduire la fonction `vanEck`.
3. Si on utilise la fonction `listeVanEck`, on obtiendra une fonction trop lente. Il faut donc utiliser la fonction `termeSuivant`.
4. Écrire une fonction intermédiaire « `listePresents(n: int) -> list[bool]` » qui renvoie une liste `P` de taille `n+1` telle que `P[i]` vaut `True` si et seulement si `i` apparaît dans `listeVanEck(n)`. Pour que la fonction soit suffisamment rapide, vous ne devez parcourir la liste `listeVanEck(n)` qu'une seule fois.
6. Écrire une fonction intermédiaire « `estDansA(k1: int ,k2: int) -> bool` » qui indique si le couple `(k1,k2)` appartient à A.