TP 10 d'informatique - MPSI - Module Tkinter - 16/01/2024

La bibliothèque "Tkinter" permet de créer des interfaces graphiques avec Python. Récupérez le fichier source disponible à l'adresse :

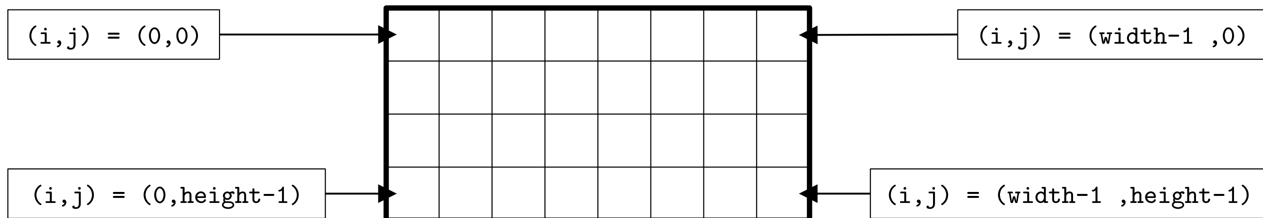
`http://informatique-lhp.fr/itc-mpsi.html`

Essayez d'exécuter le fichier pour vérifier que Tkinter fonctionne sur votre ordinateur. Si ce n'est pas le cas, réessayez après avoir lancé la commande suivante dans la console (nécessite une connexion internet) :

```
pip install tk
```

Si ça ne fonctionne toujours pas, utilisez un ordinateur du lycée.

La fonction `open_window` définie dans le fichier permet d'ouvrir une nouvelle fenêtre graphique. Elle renvoie un ***canevas***, c'est à dire une zone de la fenêtre permettant d'afficher des dessins. Un canevas est composé de pixels répartis sur une grille composée de `width` colonnes et `height` lignes. Chaque pixel est repéré par un couple `(i, j)` avec $0 \leqslant$ `i` $\leqslant$ `width - 1` et $0 \leqslant$ `j` $\leqslant$ `height - 1` où `i` est le numéro de la colonne et `j` est le numéro de la ligne. La colonne de gauche correspond à `i = 0` et celle de droite à `i = width - 1`. La ligne du haut correspond à `j = 0`, et celle du bas à `j = height - 1`.



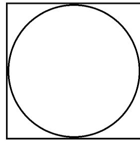
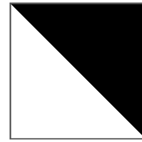
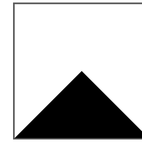
Voici les commandes dont vous aurez besoin pendant ce TP :

- `can = open_window(width, height, title)` ouvre une fenêtre graphique où :
  - `width` de type `int` est le nombre de pixels dans la largeur du canevas.
  - `height` de type `int` est le nombre de pixels dans la hauteur du canevas.
  - `title` de type `str` est le titre de la fenêtre.
- `can.winfo_reqwidth()` renvoie le nombre de pixels dans la largeur du canevas.
- `can.winfo_reqheight()` renvoie le nombre de pixels dans la hauteur du canevas.
- `can.config(bg = color)` change la couleur de l'arrière plan.
- `can.create_line(p1, p2, fill = color)` trace une ligne entre les pixels de coordonnées `p1` et `p2`. Les paramètres `p1` et `p2` sont des couples d'entiers. L'argument optionnel `fill` sert à changer la couleur de la ligne.
- `can.create_polygon(p1, p2, p3, ... , fill = color1, outline = color2)` trace un polygone défini par les pixels donnés en paramètre. Les arguments optionnels `fill` et `outline` servent à changer la couleur de l'intérieur et du bord du polygone.
- `can.create_oval(p1, p2 , fill = color1, outline = color2)` trace un ovale inscrit dans le rectangle dont `p1` est le pixel en haut à gauche et `p2` est le pixel en bas à droite.
- `afficher_pixel(can, p, fill = color)` change la couleur du pixel `p`.
- `can.update()` affiche sur l'écran les changements effectués par les appels aux fonctions précédentes.
- `can.mainloop()` lance une boucle qui attend que l'on ferme la fenêtre.

Avant de commencer les exercices, assurez vous que vous comprenez toutes les lignes de la fonction `test`.

Exercice 1.

1. Dans une fenêtre de taille 300×300, afficher un carré de 100 pixels de côté et centré au milieu. Vérifier que la console n'affiche pas `None`.
2. Afficher les formes ci-dessous dans une fenêtre de taille 401×401 (les carrés représentent la fenêtre graphique, vous n'avez pas besoin de les tracer) :

On vérifiera que les formes sont bien symétriques, en particulier sur les bords et les coins de la fenêtre graphique. On vérifiera également que la console n'affiche pas `None`.

Exercice 2. Triangle de Sierpiński

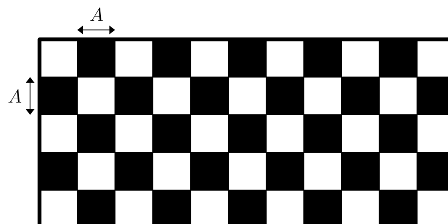
Ouvrir une fenêtre graphique et afficher en noir les pixels de coordonnées (i, j) tels que $\binom{i}{j}$ est impair (il s'agit du coefficient binomial "j parmi i"). On affichera la figure petit à petit avec une animation. Pour cela, on exécutera la commande `can.update()` à chaque fois qu'une colonne de pixels a été affichée. Tester pour une fenêtre de taille $2^7 \times 2^7$ et $2^8 \times 2^8$ et vérifier que la console n'affiche pas `None`.

Exercice 3. Damier

Écrire une fonction qui prend en entrée `width`, `height` ainsi qu'un entier $A \in \mathbb{N}^*$, et affiche un damier dont chaque case fait $A \times A$ pixels. Tester lorsque :

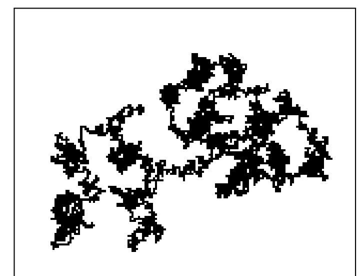
`width = 200`, `height = 300` et $A = 20$, `width = 216`, `height = 174` et $A = 47$.

Vérifier que la console n'affiche pas `None`.
Indication : on pourra d'abord chercher la condition sur le couple (i, j) pour que le pixel de coordonnées (i, j) soit noir.



Exercice 4. Marche aléatoire

Une particule est initialement placée au centre de la fenêtre graphique et se déplace n fois de suite aléatoirement d'une case dans l'une des quatre directions (haut, bas, gauche, droite). Écrire une fonction `marche_aleatoire` qui prend en entrée les dimensions `width` et `height` du canevas ainsi que n, et trace le chemin suivi par la particule. On fera en sorte d'obtenir une animation en exécutant la commande `can.update()` à chaque fois que la particule se déplace. Vérifier que la console n'affiche pas `None`.



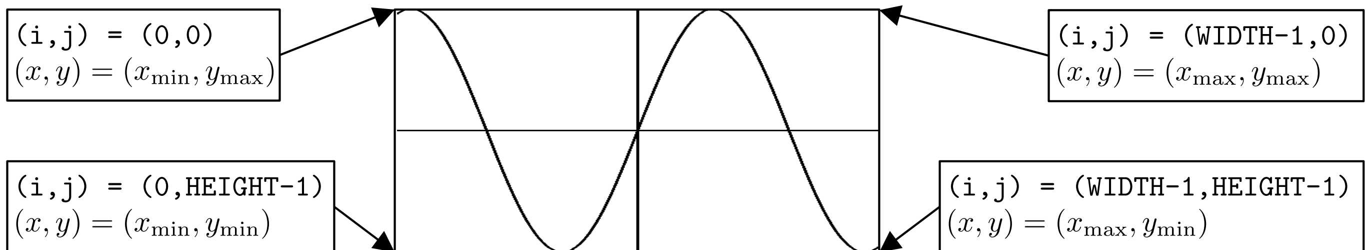
Exercice 5. Graphe d'une fonction

Les courbes que nous allons tracer viennent du site :

`https://mathcurve.com/`

Dans cet exercice, le canevas `can` représente l'ensemble $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ où $x_{\min}$, $x_{\max}$, $y_{\min}$ et $y_{\max}$ sont quatre réels. Afin de limiter le nombre d'arguments dans les fonctions, on suppose que ces quatre réels ainsi que la hauteur et la largeur de la fenêtre graphique sont définis par des variables globales. Il est donc possible d'accéder à ces six valeurs dans une fonction même si elles ne lui sont pas données en argument.

```
# Variables globales
WIDTH = 500; HEIGHT = 300
X_MIN = -1; X_MAX = 2
Y_MIN = -5; Y_MAX = 1
```



1. (a) Écrire une fonction `x_to_i` qui prend en entrée un réel x et renvoie la première coordonnée `i` des pixels dont l'abscisse est x.
   (b) Écrire une fonction `y_to_j` qui prend en entrée un réel y et renvoie la deuxième coordonnée `j` des pixels dont l'ordonnée est y.
   (c) Écrire une fonction `i_to_x` qui prend en entrée un entier `i` et renvoie l'abscisse x des pixels dont la première coordonnée est `i`.
   (d) Écrire une fonction `j_to_y` qui prend en entrée un entier `j` et renvoie l'ordonnée y des pixels dont la deuxième coordonnée est `j`.
2. À l'aide des fonctions précédentes, écrire une fonction `axe` qui prend en entrée un canevas et trace l'axe des ordonnées et l'axe des abscisses. Vérifiez que vous obtenez un tracé cohérent lorsque :

   `WIDTH = 500`, `HEIGHT = 300`, $x_{\min} = -1$, $x_{\max} = 1$, $y_{\min} = -1$ et $y_{\max} = 1$.
   `WIDTH = 500`, `HEIGHT = 300`, $x_{\min} = -1$, $x_{\max} = 2$, $y_{\min} = -5$ et $y_{\max} = 1$.

3. Écrire une fonction `tracer` qui prend en entrée une fonction `f` et trace le graphique de la fonction `f`. Testez lorsque `f` est la fonction $x \mapsto x^2$, puis $x \mapsto \sin(x)$.

Coordonnées polaires. Soit $r : \mathbb{R} \to \mathbb{R}_+$ une fonction. Le graphique de r en coordonnées polaires est l'ensemble des points M du plan tels qu'il existe $\theta \in \mathbb{R}$ avec :

$$\begin{cases} \|\overrightarrow{OM}\| = r(\theta). \\ \theta \text{ est la mesure de l'angle orienté entre } \vec{i} \text{ et } \overrightarrow{OM}. \end{cases}$$

4. Écrire une fonction `tracer_polaire` qui prend en entrée la fonction r ainsi que deux réels θ_1, θ_2, et trace le graphique en coordonnées polaires de r pour $\theta \in [\,\theta_1\,;\theta_2\,]$.
5. En ajustant les paramètres $x_{\min}, y_{\min}, x_{\max}, y_{\max}, \theta_1$ et θ_2 :
   (a) Tester la fonction `tracer_polaire` avec :

$$r : \theta \mapsto 1 + \cos(\theta) \qquad r : \theta \mapsto 1 + \cos(2\theta) \qquad r : \theta \mapsto 1 + \cos(3\theta)$$

$$r : \theta \mapsto 1 + \cos(4\theta) \qquad r : \theta \mapsto 1 + \cos\left(\frac{7}{2}\theta\right)$$

On exploitera le fait que les fonctions sont périodiques et on fera en sorte de n'afficher chaque point de la courbe qu'une seule fois.

Les courbes obtenues s'appellent des épicycloïdes.

(b) Tester la fonction `tracer_polaire` avec :

$$r : \theta \mapsto \frac{1}{2 + e^{\theta/10}} \qquad \text{et} \qquad \theta_1 = -\theta_2$$

La courbe obtenue s'appelle la courbe du ressort.

(c) Tester la fonction `tracer_polaire` avec :

$$r : \theta \mapsto \frac{1}{\sqrt{\theta}}$$

La courbe obtenue est la moitié d'un lituus.

Courbes paramétrées. Soient $f_x : \mathbb{R} \to \mathbb{R}$ et $f_y : \mathbb{R} \to \mathbb{R}$ deux fonctions. La courbe paramétrée associée à f_x et f_y est l'ensemble des points du plan de coordonnées $(f_x(t), f_y(t))$ où $t \in \mathbb{R}$.

6. Écrire une fonction `tracer_param` qui prend en entrée les fonctions f_x, f_y ainsi que deux réels t_1, t_2, et trace la courbe paramétrée associée à f_x et f_y pour $t \in [\,t_1\,;t_2\,]$.

7. En ajustant les paramètres $x_{\min}, y_{\min}, x_{\max}, y_{\max}, t_1$ et t_2 :

   (a) Tester la fonction `tracer_param` avec :

   $$f_x : t \mapsto e^{-t} \qquad\qquad f_y : t \mapsto \cos(e^t)$$

   La courbe obtenue correspond au graphe de $x \mapsto \cos(1/x)$ pour $x > 0$ (fonction bornée sans limite en 0^+).

   (b) Tester la fonction `tracer_param` avec :

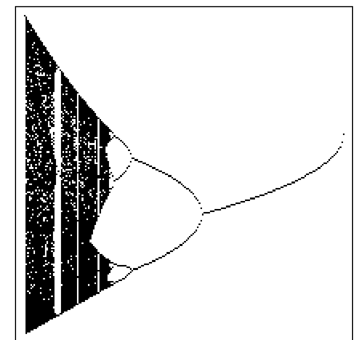
   $$f_x : t \mapsto \frac{t}{\sqrt{|t|}}\cos(|t|) \qquad\qquad f_y : t \mapsto \frac{t}{\sqrt{|t|}}\sin(|t|) \qquad\qquad t_1 = -t_2$$

   La courbe obtenue s'appelle la spirale de Fermat.

Exercice 6. Diagramme de bifurcation

Dans cet exercice, le canevas représente l'ensemble $[-2, 0.25] \times [-2, 2]$. Pour tout $x \in [-2, 0.25]$, on définit la suite $(y_k)_{k\in\mathbb{N}}$ par :

$$\begin{cases} y_0 = 0 \\ y_{k+1} = y_k^2 + x \qquad \text{pour tout } k \geqslant 0. \end{cases}$$



Écrire une fonction `dessiner_DB` (sans argument) qui affiche tous les points (x, y_k) où $x \in [-2, 0.25]$ et $k \in [\![1000, 1500]\!]$. On pourra définir des variables globales comme dans l'exercice 5 et utiliser les fonctions de la question 1 de l'exercice 5.

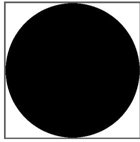
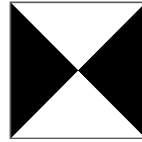
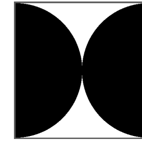
Si Tkinter ne fonctionne pas sur votre PC, indiquez le au début de votre fichier source et faites les exercices sans les tester.

Exercice 7.

Écrire trois fonctions sans argument :

`dessin5() -> Nonetype`   `dessin6() -> Nonetype`   `dessin7() -> Nonetype`

qui dessinent les formes ci-dessous dans une fenêtre de taille 301×301 (les carrés représentent la fenêtre graphique, vous n'avez pas besoin de les tracer) :

On vérifiera que les formes sont bien symétriques, en particulier sur les bords, les coins et le centre de la fenêtre graphique. La console ne doit pas afficher `None`.

Exercice 8. Automates cellulaires

Un automate cellulaire est un système dynamique régi par des règles locales. Dans cet exercice, on s'intéresse aux automates cellulaires en dimension 1. Une ***configuration*** est une ligne composée de `width` cellules, chaque cellule étant blanche ou noire. En Python, une configuration sera représentée par une liste de taille `width` contenant des `0` (pour les cellules blanches) et des `1` (pour les cellules noires). Par exemple, la liste `[0,1,0,0,0,1,1,0,1,0,1,1,1]` correspond à la configuration :

Le système évolue du temps `t = 0` au temps `t = height - 1`. À chaque instant, chaque cellule est mise à jour en fonction de la couleur de ses deux voisines et de sa propre couleur. Pour cela, on dispose d'un dictionnaire `regles` tel que :

- Les clés de `regles` sont tous les triplets `(c1, c2, c3)` $\in$ `{0, 1}`^{3}.
- Les valeurs de `regles` sont des `0` et des `1`.

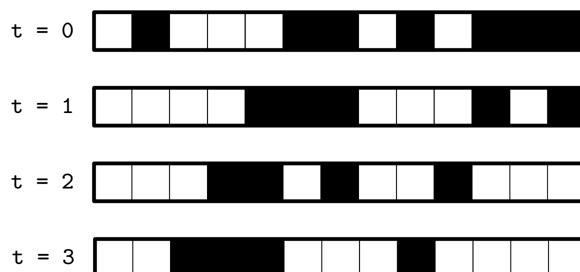
Soit `i` $\in [\![$`0, width - 1`$]\!]$ l'indice d'une cellule et soit `(c1, c2, c3)` $\in$ `{0, 1}`^{3}. On suppose qu'au temps `t`, le voisin gauche de la cellule `i` est de couleur `c1`, la cellule `i` est de couleur `c2` et le voisin droit de la cellule `i` est de couleur `c3`. Alors au temps `t+1`, la cellule `i` sera de couleur `regles[(c1,c2,c3)]`. On considère que la voisine gauche de la cellule d'indice `0` est la cellule d'indice `width-1` et que la voisine droite de la cellule d'indice `width-1` est la cellule d'indice `0`.

Par exemple, si au temps `t = 0`, on a la configuration dessinée ci-dessus alors au temps `t = 1` :

- La cellule d'indice `0` est de couleur `regles[(1,0,1)]`.
- La cellule d'indice `1` est de couleur `regles[(0,1,0)]`.
- La cellule d'indice `2` est de couleur `regles[(1,0,0)]`.
- La cellule d'indice `3` est de couleur `regles[(0,0,0)]` ...

Voici un exemple de règles et l'évolution associée :

```
regles = {
    (0,0,0): 0, (0,0,1): 1,
    (0,1,0): 0, (0,1,1): 1,
    (1,0,0): 0, (1,0,1): 0,
    (1,1,0): 1, (1,1,1): 0,
}
```



1. Écrire une fonction

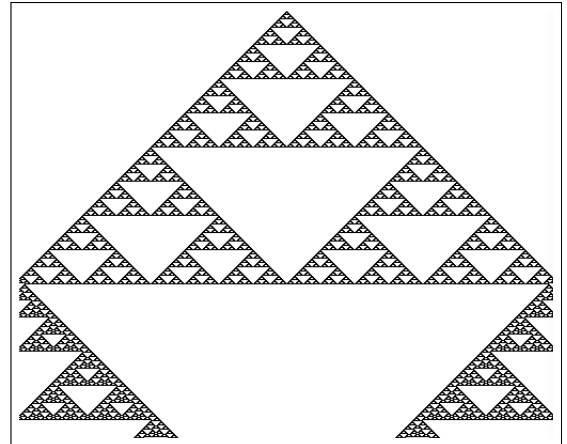
```
etape(config: list[int], regles: dict[int*int*int, int]) -> list[int]
```

qui prend en entrée une configuration à un instant `t` ainsi qu'un dictionnaire `regles`, et renvoie la configuration à l'instant `t+1`.

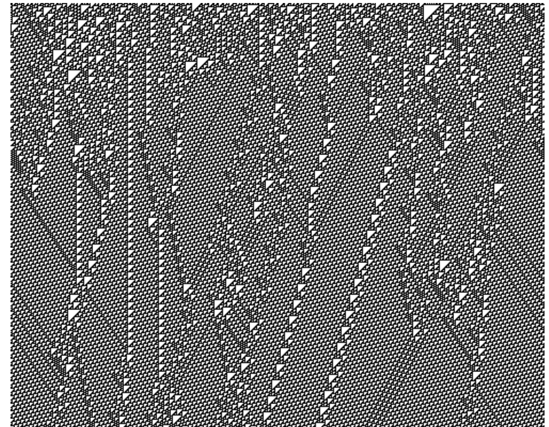
Le ***diagramme espace-temps*** d'un automate cellulaire est une grille avec `width` colonnes et `height` lignes où pour tout $\texttt{i} \in [\![0, \texttt{width - 1}]\!]$ et tout $\texttt{t} \in [\![0, \texttt{height - 1}]\!]$, la pixel d'indice `(i,t)` est de la couleur de la cellule numéro `i` au temps `t`.

2. Écrire une fonction `diagramme` qui prend en entrée une configuration initiale, le dictionnaire `regles` ainsi que l'entier `height`, et affiche le diagramme espace-temps associé. Voici deux exemples et les diagrammes espace-temps obtenus.

```
width = 500; height = 400
config_ini = [0]*width
config_ini[width//2] = 1
regles = {
    (0,0,0):0, (0,0,1):1,
    (0,1,0):1, (0,1,1):1,
    (1,0,0):1, (1,0,1):1,
    (1,1,0):1, (1,1,1):0,
}
diagramme(config_ini, regles, height)
```



```
width = 500; height = 400
config_ini = [random.randint(0,1)
              for _ in range(width)]
regles = {
    (0,0,0):0, (0,0,1):1,
    (0,1,0):1, (0,1,1):1,
    (1,0,0):0, (1,0,1):1,
    (1,1,0):1, (1,1,1):0,
}
diagramme(config_ini, regles, height)
```



Exercice 9. Fourmi de Langton (exercice facultatif)

Une fourmi se déplace sur la fenêtre graphique en faisant chaque pas vers le haut, vers le bas, vers la droite ou vers la gauche. Elle suit les règles suivantes :

- Si elle est sur un pixel noir, alors elle change la couleur du pixel en blanc, tourne de 90° vers la gauche et avance d'un pas.
- Si elle est sur un pixel blanc, alors elle change la couleur du pixel en noir, tourne de 90° vers la droite et avance d'un pas.

Écrire une fonction de signature « `fourmi(width: int, height: int) -> NoneType` » qui prend en entrée la largeur et la hauteur du canevas et affiche l'évolution de la fenêtre graphique lorsque la fourmi se déplace. On supposera que la fenêtre est périodique (lorsque la fourmi sort de l'un des bords, elle apparaît sur le bord opposé). Vous pouvez choisir arbitrairement la position initiale, la direction initiale et les couleurs initiales des pixels. Essayez avec différentes tailles de fenêtre, en particulier lorsque la fenêtre n'est pas carrée.