TP 7 d'informatique - MPSI - Module Tkinter - 02/12/2025

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

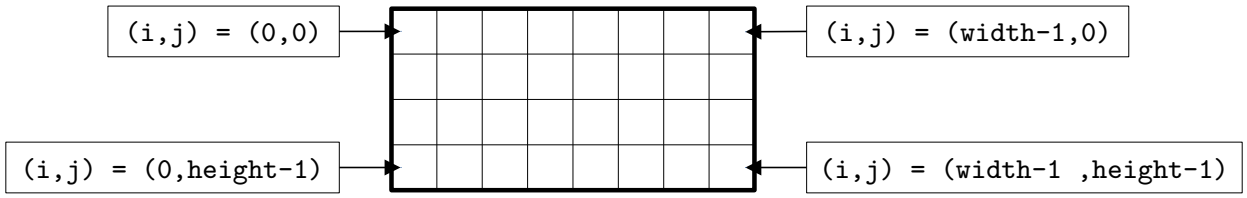
http://informatique-lhp.fr/itc-mpsi.html

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`.

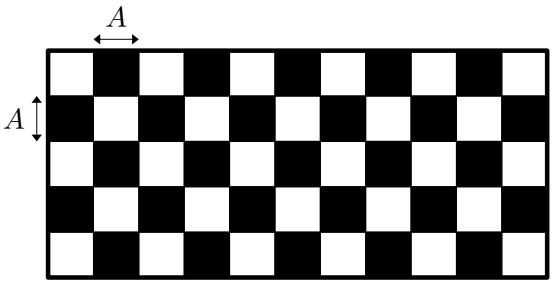
Exercice 3. Damier

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 une condition nécessaire et suffisante sur le couple (i, j) pour que le pixel de coordonnées (i, j) soit noir.



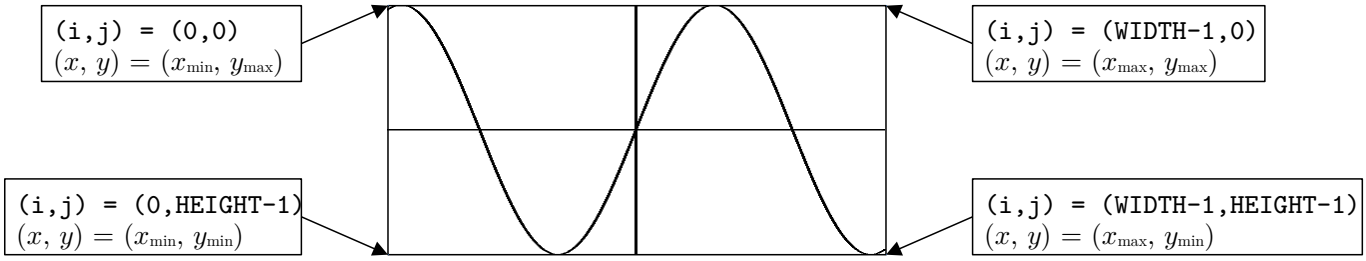
Exercice 4. Graphe d'une fonction

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 $f_1 : \mathbb{R} \to \mathbb{R}$ et $f_2 : \mathbb{R} \to \mathbb{R}$ deux fonctions. La courbe paramétrée associée à f_1 et f_2 est l'ensemble des points du plan de coordonnées $(f_1(t), f_2(t))$ où $t \in \mathbb{R}$.

6. Écrire une fonction `tracer_param` qui prend en entrée les fonctions f_1, f_2 ainsi que deux réels t_1, t_2, et trace la courbe paramétrée associée à f_1 et f_2 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 :

   $$f_1 : t \mapsto e^{-t} \qquad\qquad f_2 : t \mapsto \cos(e^t)$$

   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 :

   $$f_1 : t \mapsto \frac{t}{\sqrt{|t|}} \cos(|t|) \qquad f_2 : t \mapsto \frac{t}{\sqrt{|t|}} \sin(|t|) \qquad t_1 = -t_2$$

   La courbe obtenue s'appelle la spirale de Fermat.

Exercice 5. Fourmi de Langton

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.

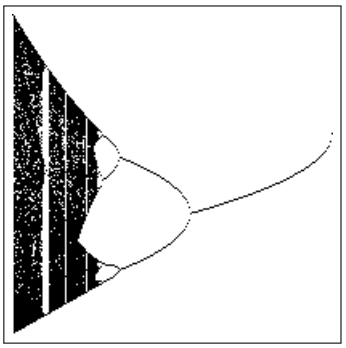
Écrire une fonction de signature « `fourmi(width: int, height: int) -> NoneType` » qui prend en entrée la largeur et la hauteur du canevas et affiche l'évolution de la fenêtre graphique lorsque la fourmi se déplace. On supposera que la fenêtre est périodique (lorsque la fourmi sort de l'un des bords, 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.

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 4 et utiliser les fonctions de la question 1 de l'exercice 4.



Si Tkinter ne fonctionne pas sur votre PC, indiquez le au début de votre fichier source et faites l'exercice 7 sans le 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. Recherche du second maximum dans une liste

Soit `L` une liste d'entiers. On souhaite trouver le second maximum de `L`. Par exemple, le second maximum de `[1,5,2]` est `2`, le second maximum de `[1,5,5]` est `5` et le second maximum de `[2,6,4,6]` est `6`. En particulier, si le maximum apparaît plusieurs fois dans `L`, on considère que le second maximum est égal au maximum.

1. Écrire une fonction « `snd_max(L: list[int]) -> int` » qui renvoie le second maximum de `L`. Dans le cas où il n'y a pas de second maximum, votre fonction déclenchera une erreur.

Exercice 9. Nombres opposés

Écrire une fonction « `nb_opp(L: list[int]) -> int` » qui prend en entrée une liste `L` d'entiers non nuls et renvoie le nombre d'entiers strictement positifs x tels que x et $-x$ apparaissent dans `L`. Par exemple, la fonction renvoie `2` pour la liste `[-4,5,3,2,4,4,-2,5,-4]`. En effet, les entiers strictement positifs `2` et `4` ainsi que leurs opposés apparaissent dans la liste. **La complexité de votre fonction devra être linéaire en la taille de `L`.** Si vous n'y arrivez pas, vous pouvez lire les indications qui suivent.

Indications (essayez de faire l'exercice sans lire ce qui suit). On pourra créer un dictionnaire `d` dont l'ensemble des clés est égal à l'ensemble des éléments des `L` (les valeurs de `d` sont quelconques). Ainsi, grâce au dictionnaire `d`, pour chaque élément `e` de `L`, on pourra tester en temps constant si `-e` est un élément de `L`. Pensez à organiser votre code avec des fonctions intermédiaires.

Exercice 10. Rallonges audio (exercice facultatif)

Cet exercice est issu du site France-ioi :

http://www.france-ioi.org/algo/task.php?idChapter=761&idTask=874

Vous souhaitez raccorder votre nouvel ordinateur à l'amplificateur de votre super chaîne hi-fi afin de pouvoir mettre à fond l'album "Wish You Were Here" (du groupe Pink Floyd). Vous espérez que la mélodie des guitares électriques fera venir en vous l'inspiration nécessaire pour résoudre des sujets d'algorithmique.

En même temps, vous souhaitez vous rapprocher le plus possible de la borne wifi par laquelle vous accédez à Internet afin de pouvoir télécharger tous les autres albums des Pink Floyd avec un débit maximal (téléchargé légalement, bien sûr). Cette borne wifi est commune à votre immeuble et peut théoriquement

vous fournir une très bonne bande passante. Malheureusement, elle se situe un peu loin de chez vous et cela est un facteur limitant votre débit. Vous avez déjà remarqué qu'il vous était possible d'améliorer le débit rien qu'en vous déplaçant de quelques mètres en direction de la borne wifi.

Votre but va donc être de combiner les rallonges audio dont vous disposez afin d'obtenir un maximum de longueur, et pouvoir ainsi écouter de la musique tout en étant aussi près que possible de la borne wifi.

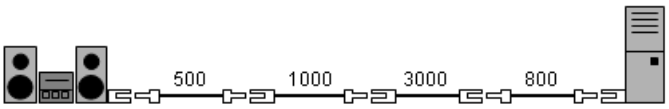
Vous avez à votre disposition un certain nombre de rallonges. Chaque rallonge est caractérisée par sa longueur, et par la nature des connecteurs à ses deux extrémités : chaque connecteur peut être soit mâle, soit femelle. Remarque technique : on ne peut pas connecter des fiches mâles entre elles, ni deux fiches femelles entre elles. Notez enfin que la carte son de votre ordinateur ainsi que l'amplificateur de votre chaîne hi-fi portent tous deux des connecteurs femelles (il faut donc arriver dessus avec des connecteurs mâles).

Entrée. Votre fonction `dist_max` prend en entrée une liste dont chaque élément est un triplet (C_1, C_2, L). C_1 et C_2 décrivent les connecteurs (1 pour mâle, 0 pour femelle), et L donne la longueur de la rallonge. Notez que le signal peut passer dans une rallonge dans le sens que l'on veut. Ainsi le signal peut traverser la rallonge de C_1 vers C_2 ou de C_2 vers C_1.

Sortie. Vous devez renvoyer un seul entier : la distance maximale à laquelle vous pouvez placer votre ordinateur de la chaîne hi-fi tout en lui envoyant le signal musical à travers les rallonges. S'il est impossible de relier les deux appareils avec les rallonges dont vous disposez, renvoyez -1.

Exemple. Entrée : `[(1,0,1000), (1,1,500), (0,0,2000), (1,1,800), (0,0,3000)]`
Sortie : `5300`

Commentaires. Utilisez la séquence suivante par exemple :



Notez que la seconde rallonge a été retournée par rapport à sa description dans l'entrée. La longueur totale obtenue est `500 + 1000 + 3000 + 800 = 5300`. Vous pouvez utiliser la commande `L.sort()` qui permet de trier une liste `L` et la commande `sum(L)` qui renvoie la somme des éléments de `L`.