université angers
FACULTÉ DES SCIENCES
Unité de formation et de recherche

Département Informatique
L1 Informatique
Jean-Michel Richer
Développement Python
2024/2025

jean-michel.richer@univ-angers.fr

Contrôle Continu L1 Python Juin 2025

Vous avez le droit de consulter mon site web ou la documentation Python ainsi que vos exercices de TP au **format papier**. Il est interdit de faire appel à des outils comme ChatGPT. Tout programme qui ne s'exécute pas correctement (erreur, exception) est compté pour 0.

Vous devez rendre quatre fichiers qui correspondent chacun à une question. Si vous ne répondez pas à une question ne rendez pas le fichier correspondant.

Exercice 1 - 5 pts -

Ecrire un programme `chaos.py` qui dessine avec matplotlib une suite chaotique. On créera une fonction `chaos(x,r)→list` qui génère une liste de 100 valeurs définie par :

$$x_{n+1} = r \times x_n \times (1 - x_n)$$

On prendra $x_0 = 0.1$ (qui doit faire partie de la liste) et $r = 3.69$ et on dessinera cette liste :

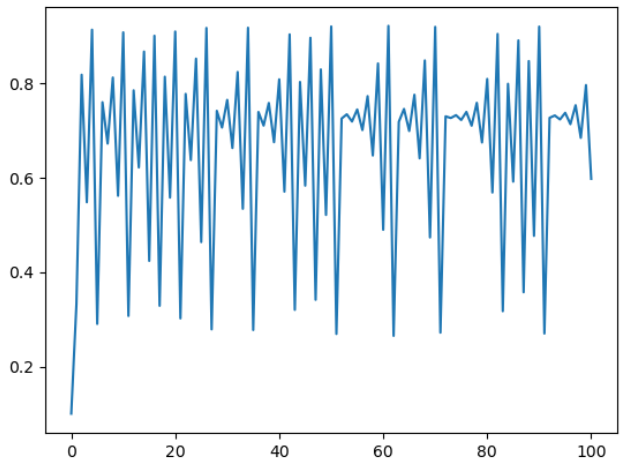

FIGURE 1 – Exemple de résultat attendu

Exercice 2 - 5 pts -

Ecrire un programme **creer_lab.py** qui crée un labyrinthe de taille $N \times N$ sous forme d'une liste de liste d'entiers, étant donnée une chaîne composée de caractères `'0'` et de `'1'`. On partira de la chaîne **description** :

```
description = "1100101011100111"
```

Créer la variable **labyrinthe** en tant que liste de liste d'entiers. On affichera donc en résultat le contenu de **labyrinthe**, puis sous forme matricielle :

```
# affichage de la variable labyrinthe (liste de liste)
        [[1, 1, 0, 0], [1, 0, 1, 0], [1, 1, 1, 0], [0, 1, 1, 1]]
# affichage sous forme matricielle
        1 1 0 0
        1 0 1 0
        1 1 1 0
        0 1 1 1
```

Il faudra calculer la taille N du labyrinthe en fonction de la longueur l de la chaîne, celle-ci doit être égale à un carré (9, 16, 25, etc) et donc $N = \sqrt{l}$.

Exercice 3 - 5 pts -

Ecrire un programme `dessine_chemin.py` qui dessine avec matplotlib le chemin correspondant au déplacement dans un labyrinthe de taille 4×4.

A partir du chemin suivant qui correspond à des coordonnées au format (y, x) :

```
chemin = [(0, 0), (1, 0), (1, 1), (1, 2), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3)]
```

On veut obtenir le dessin ci-après (voir Figure 2) sachant que :

- les coordonnées du chemin pour $x = 0, 1, 2, 3$ sont transformées respectivement en $x = 1, 2, 3, 4$ sur le dessin
- les coordonnées du chemin pour $y = 0, 1, 2, 3$ sont transformées respectivement en $y = 4, 3, 2, 1$ sur le dessin
- on affichera un cercle rouge pour chaque point du chemin
- on affichera une ligne (en bleu) entre deux points

Il suffit d'utiliser la fonction `plot` et la fonction `scatter` avec les bon paramètres.

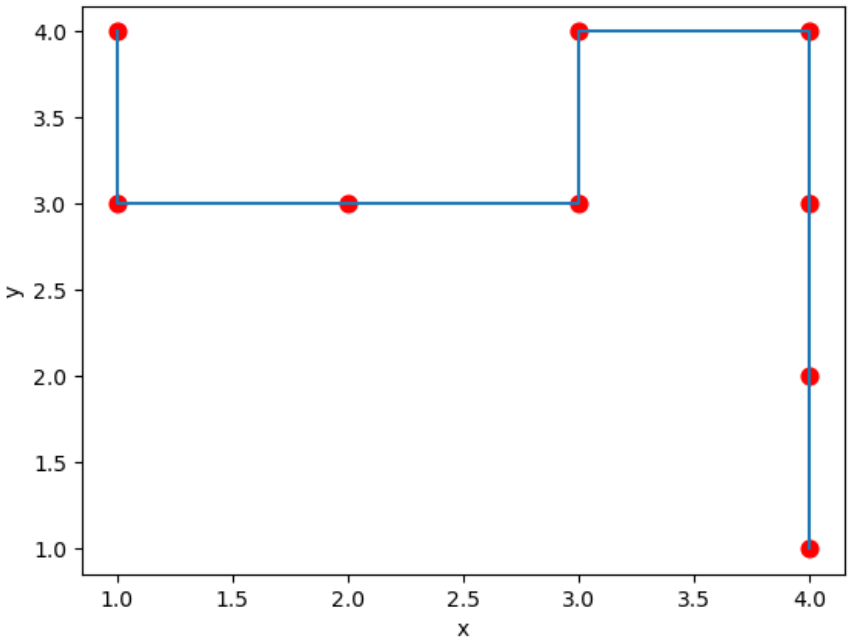

FIGURE 2 – Exemple de résultat attendu

Exercice 4 - 5 pts -

Ecrire un programme **labyrinthe.py** qui, partant du point de coordonnées $(0,0)$, permet d'atteindre (si cela est possible) le point (3,3). Le labyrinthe est représenté sous la forme d'une liste de listes d'entiers composées de 0 (pas de chemin) ou de 1 (chemin). On utilisera :

```
labyrinthe = [
    [1, 0, 1, 1],
    [1, 1, 1, 1],
    [0, 1, 0, 1],
    [1, 1, 1, 1]
]
```

1. **labyrinthe** est une variable globale ainsi que **chemins_valides** qui sera initialisée comme une liste vide, elle contiendra les chemins trouvés
2. définir une procédure récursive **explorer(lab,dim,chemin,yd,xd,yf,xf)→None** qui prend en paramètres un labyrinthe, sa dimension (ici 4), le chemin courant ainsi que les coordonnées de début du chemin (y_d, x_d) et de fin de chemin (y_f, x_f), c'est à dire la position à atteindre
3. dans cette fonction
   - la condition d'arrêt correspond à $x_d = x_f$ et $y_d = y_f$, dans ce cas on créera une copie du chemin courant (`chemin.copy()`) que l'on ajoutera à la liste des **chemins_valides**, puis on sortira de la procédure (`return`)
   - la propriété de récurrence consiste à se déplacer vers le haut, le bas, la droite, la gauche quand cela est possible, c'est à dire que les nouvelles coordonnées (y, x) doivent rester dans l'intervalle $[0, 3]$ et **lab[y][x]** doit être égal à 1
   - dans ce cas on ajoutera la nouvelle position (y, x) vers laquelle on va se déplacer au **chemin** courant, puis on relancera l'exploration
   - on n'oubliera pas de supprimer (y, x) de **chemin** si on doit tester une autre possibilité de déplacement
4. on réalisera l'exploration avec l'appel suivant :
   `explorer(labyrinthe, 4, [(0,0)], 0, 0, 3, 3)`

Vous devriez obtenir les trois chemins suivants (cf. Fig. 3, page suivante) et les afficher de manière lisible comme suit (l'ordre d'apparition des chemins n'est pas important) :

```
[(0, 0), (1, 0), (1, 1), (1, 2), (1, 3), (2, 3), (3, 3)]
[(0, 0), (1, 0), (1, 1), (1, 2), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3)]
[(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3)]
```

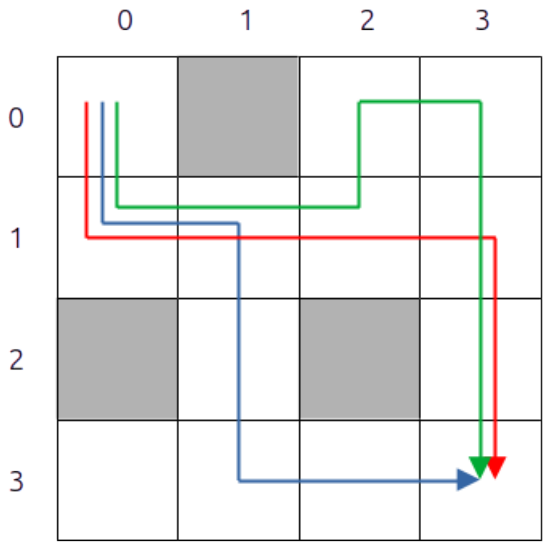

FIGURE 3 – Exemples de chemins de $(0, 0)$ vers $(3, 3)$