TP n°3 : quelques algorithmes classiques sur les listes

1 Comment construire des listes ?

Exercice 1 Construire la liste des multiples de 3 ou de 5 strictement inférieurs à 1000. On pourra utiliser deux méthodes, l'une étant à l'aide d'une «comprehension list».

Exercice 2 Écrire une fonction `liste_diviseurs(n)` qui renvoie la liste des diviseurs d'un entier n.

2 Quelques fonctions statistiques

Exercice 3 (maximum)

1. Écrire une fonction `maximum(t)` qui renvoie le plus grand élément d'un tableau t.
2. Écrire une fonction `Maximum(t)` qui renvoie le plus grand élément d'un tableau t ainsi que son indice (s'il y a plusieurs éléments qui réalisent le maximum, on renverra l'indice de l'un de ces éléments maximum). Par si $t = [40, 50, 80, 12]$, 80 est le plus grand élément de t, son indice est 2. `Maximum(t)` renverra le tuple `(80,2)`.

Exercice 4 (Test de croissance) Écrire une fonction booléenne `estCroissante(t)` qui prend en argument une liste t de nombres, et renvoie `True` si les éléments de t sont classés par ordre croissant et renvoie `False` sinon.

Exercice 5 (Moyenne et variance)

1. Écrire une fonction `moyenne` qui calcule la moyenne des éléments d'un tableau.
2. La variance d'une liste est la moyenne des carrés des écarts à la moyenne. On propose la fonction suivante :

```
def variance(t):
    n = len(t)
    s = 0
    for x in t:
        s += (x - moyenne(t))**2
    return s/n
```

Combien d'additions sont effectuées lorsque l'on exécute `variance(t)` avec une liste t de longueur n. Comment améliorer l'efficacité ?

Exercice 6 (Écart extrême)

1. Écrire une fonction `ecartMinimum` qui prend en argument un tableau t non vide d'entiers et qui renvoie le minimum des valeurs $|t_i - t_j|$ lorsque $i > j$ (en Python la valeur absolue est implémentée par la fonction `abs`). Par exemple, si $t = [6, 4, 7, 10]$, `ecartMinimum(t)` renverra 1 car $|7 - 6| = 1$.

2. Déterminer le nombre de comparaisons effectuées par `ecartMinimum(t)` lorsque le tableau `t` a pour longueur n.

3. Écrire une fonction `ecartMaximum` qui prend en argument un tableau t non vide d'entiers et qui renvoie le maximum des valeurs $|t_i - t_j|$ lorsque $i > j$ (en Python la valeur absolue est implémentée par la fonction `abs`). Par exemple, si $t = [6, 4, 7, 10]$, `ecartMaximum(t)` renverra 6 car $|10 - 4| = 6$.

4. Déterminer la complexité de votre fonction (il est possible d'obtenir une complexité linéaire).

3 Tableau d'occurrences

Exercice 7 (Tri par dénombrement) Soit t un tableau dont tous les éléments sont des entiers naturels compris entre 0 et $k-1$ avec $k \in \mathbb{N}^*$ un entier fixé.

1. Écrire une fonction `occurrence(t, k)` qui renvoie une liste `occ` de longueur k dont le i-ème élément désigne le nombre d'occurrences de l'entier i dans la liste t.

   Par exemple, si $t = [0, 1, 1, 0, 3, 0, 3, 0]$ et $k = 4$, `occurrence(t, k)` renverra $[4, 2, 0, 2]$ car 0 est présent 4 fois dans t, 1 est présent 2 fois, 2 est absent et 3 est présent 2 fois.

2. En déduire une fonction `counting_sort(t, k)` qui trie le tableau t dont les éléments sont dans $[\![0, k-1]\!]$.

Exercice 8 (Découverte des dictionnaires) On considère une liste contenant des mots. Par exemple $t = [\text{'vache'}, \text{'brebis'}, \text{'vache'}, \text{'chien'}, \text{'chien'}, \text{'vache'}, \text{'vache'}]$.

Le mot `'vache'` apparaît 4 fois, `'brebis'` apparaît 1 fois... Si l'on désire mémoriser toutes les occurrences de cette liste de mots, la structure de liste ne semble pas idéale. En effet dans quel index va -t-on placer le mot `'vache'` ? Il existe une structure de données adaptée à cette situation : le dictionnaire. La notion d'index est remplacée par la notion de clef et ce n'est plus un entier. Voici comment on l'utilise dans le langage Python

```
occ = dict() # dictionnaire vide

for mot in t:
    if mot not in occ:
        occ[mot] = 1
    else:
        occ[mot] += 1
print(occ)
```

1. Exécuter `occ['vache']`, `occ['chien']`. Commenter.

2. Créer une fonction qui prend en argument un texte (ne contenant que des lettres majuscules et pas de signes de ponctuation) et renvoie le dictionnaire des occurrences des lettres (attention à la gestion des espaces).

3. Proposer ensuite un script qui donne la lettre la plus fréquente.

4 Graphe de suites récurrentes avec matplotlib.pyplot

On va tracer des graphiques avec le sous-module `matplotlib.pyplot`. Voici un script qui permet de retrouver la syntaxe.

```
import matplotlib.pyplot as plt

X = [-1, 0, 1, 2]  # la liste des abscisses
Y = [3, 2, 4, 1] # la liste des ordonnées

plt.plot(X, Y) # pour créer le graphique
plt.show() # pour afficher le graphique
```

Exercice 9 Représenter le graphe discret des onze premiers points de la suite récurrente u définie par $u_0 = -8.41$ et $u_{n+1} = -0.5u_n + 63$.

Exercice 10 (La suite logistique) Soit r un réel de $[0, 4]$. On considère la suite logistique définie par

$$u_{n+1} = ru_n(1 - u_n) \quad \text{et} \quad u_0 = 0.5.$$

Cette suite très célèbre permet de modéliser certaines dynamiques de population. Elle a un comportement très différent selon les valeurs de r. On se propose de tracer le nuage de points suivants.

1. Créer une subdivision uniforme `R` de l'intervalle $[0, 4]$ en 201 points.
2. Pour chaque élément r de R, représenter les points de coordonnées $(r, u_{100}), (r, u_{101}), \ldots, (r, u_{200})$.
3. Sauver votre graphique à l'aide de `plt.savefig('fonction_logistique.pdf')`.

5 Une première utilisation de la programmation dynamique

Exercice 11 (Nombres de Catalan) Les nombres de Catalan C_n sont définies par $C_0 = 1$ et

$$\forall n \in \mathbb{N}^*, \quad C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

1. Calculer «avec un papier et un crayon» C_n pour $n \leqslant 5$, on vérifiera que $C_5 = 42$.
2. Écrire une fonction **récursive** `catalan_rec` prenant en argument un entier naturel n et renvoyant la valeur de C_n. Tester là pour $n = 5$ puis $n = 10$. Commenter.
3. Autre méthode : construire un tableau de longueur 11, puis compléter ce tableau afin qu'il contienne les valeurs $C_0, C_1, \ldots, C_{10}$.
4. En déduire une nouvelle fonction `catalan` prenant en argument un entier naturel n et renvoyant la valeur de C_n.
5. Déterminer le nombre d'additions et de multiplications nécessaires pour calculer C_{10}.

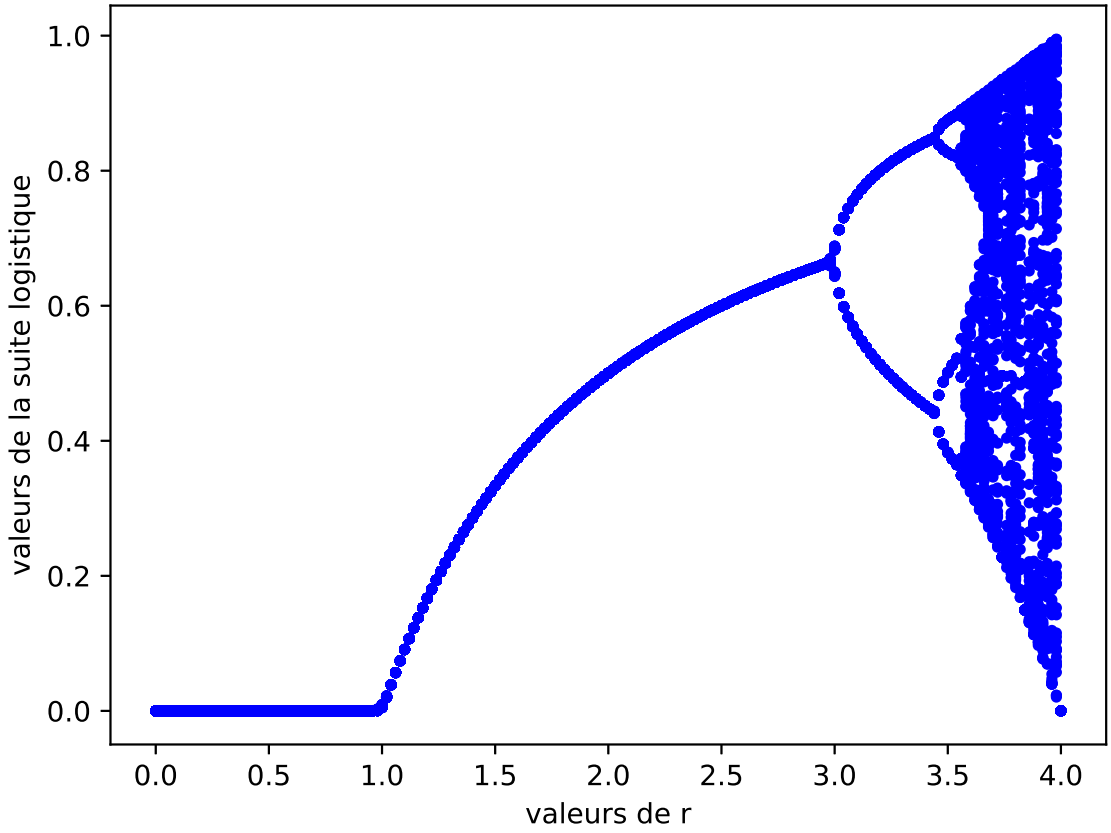

FIGURE 1 – Comportement de la suite logistique