Les suites et les listes

I. Algorithme de seuil

Python est un outil particulièrement efficace pour modéliser et étudier des suites.
En particulier, une question qui nous intéresse souvent lorsque l'on étudie une suite monotone est de savoir à partir de quel rang les termes de la suites se trouve-t-ils tous dans un intervalle donné.
C'est précisément pour répondre à cette question que l'on programme un **algorithme de seuil**, comme nous allons le voir dans ce premier exercice.
Problème du Nénuphar :
Un lac contient un nénuphar magique dont la surface double tous les jours.
Le jour 0, son aire est de 10 mm^2.
On notera a_n l'aire couverte par le nénuphar au jour n.

1) Modélisation de la suite

1. Quelle est l'aire couverte par le nénuphar au jour 1 ? au jour 2 ? au jour 3 ?

   ---

2. Montrez que la suite (a_n) peut être définie par la formule explicite $\boldsymbol{a_n = 10 \times 2^n}$.

   ---

3. Définissez dans un éditeur Python une fonction `def a(n):` renvoyant la valeur du terme de rang n de la suite a.
4. Vérifiez en utilisant votre fonction dans la console que les résultats pour les rangs 0, 1, 2 et 3 sont cohérents avec vos calculs, puis appelez-moi.
5. Quelle sera l'aire occupée par le nénuphar au jour 20 ? .....

2) Algorithme de seuil

En réalité, la surface du lac est de 10 km^2. En supposant que le nénuphar épouse parfaitement les contours du lac, on cherche à savoir au bout de combien de temps, il aura recouvert entièrement le lac.
Notez que s'il est possible de répondre algébriquement à ce genre de question, vous ne savez pas le faire pour l'instant (bientôt).

1. À l'aide d'une boucle non bornée, `while ...:`, déterminez le jour de recouvrement du lac.

   ---

2. Appelez-moi pour vérifier puis notez ci-dessous la structure de l'algorithme utilisé, c'est un **algorithme de seuil**.

3. Modifier votre dernier code pour l'inclure dans une fonction `def seuil(N):` renvoyant le premier rang n_0 pour lequel la suite (a_n) dépasse le seuil N.
4. Vérifiez en utilisant votre fonction dans la console qu'il faut bien 54 jours pour dépasser 123456789123456789 mm^2 puis appelez-moi.

II. Liste et paradoxe de Zénon

Le paradoxe de Zénon (version caillou)

« Zénon se tient à huit mètres d'un arbre, tenant une pierre. Il lance sa pierre dans la direction de l'arbre. Avant que le caillou puisse atteindre l'arbre, il doit traverser la première moitié des huit mètres. Il faut un certain temps, non nul, à cette pierre pour se déplacer sur cette distance. Ensuite, il lui reste encore quatre mètres à parcourir, dont elle accomplit d'abord la moitié, deux mètres, ce qui lui prend un certain temps. Puis la pierre avance d'un mètre de plus, progresse après d'un demi-mètre et encore d'un quart, et ainsi de suite *ad infinitum* et à chaque fois avec un temps non nul. Zénon en conclut que la pierre ne pourra pas frapper l'arbre, puisqu'il faudrait pour cela que soit franchie effectivement une série infinie d'étapes, ce qui est impossible.
Le paradoxe se résout en soutenant que le mouvement est continu; le fait qu'il soit divisible à l'infini ne le rend pas impossible pour autant. De plus, en analyse moderne, le paradoxe est résolu en utilisant fondamentalement le fait qu'une somme infinie de nombres strictement positifs peut converger vers un résultat fini. »

Source : *Wikipedia*

En effet, il est possible de montrer qu'en additionnant ainsi toutes les subdivisions parcourues, après une infinité d'étapes, la suite d_n (de la distance totale parcourue) converge bien vers huit mètres.

$$\lim_{n\to+\infty} d_n = 8$$

Nous allons nous intéresser ici à la visualisation de cette limite.

1) Modélisation de la suite

1. Quelle est la distance parcourue par le caillou au bout de 1 étape? 2 étapes? 3 étapes?

   ---

2. Définissez dans un éditeur Python une fonction `def d(n):` renvoyant la valeur du terme de rang n de la suite d. Vous aurez besoin pour cela d'utiliser une boucle bornée : `for k in range(1,n+1):` .
3. Vérifiez en utilisant votre fonction dans la console que les résultats pour les rangs 1, 2 et 3 sont cohérents avec vos calculs, puis appelez-moi.
4. Quelle sera la distance parcourue après 30 étapes? .....

2) Les listes

Pour pouvoir conjecturer une limite, nous aurions besoin de pouvoir consulter tous les premiers termes de la suite (d_n) en un seul coup d'œil.
Pour ce faire, nous allons utiliser un nouvel objet : **les listes**.

- Une liste est une structure de données **indicées** qui contient une série de valeurs. C'est l'équivalent Python d'une suite.
- On accède à l'élément n de la liste u en saisissant `u[n]` .
- Une liste est déclarée par une série de valeurs séparées par des **virgules**, et le tout encadré par des **crochets**.

1. Directement dans la console Python, définissez les trois listes suivantes, en copiant le code ci-dessous :

```
>>> animaux = ["girafe", "tigre", "singe", "souris"]
>>> tailles = [5, 2.5, 1.75, 0.15]
>>> mixte = ["girafe", 5, "souris", 0.15]
```

2. Vérifiez que les commandes suivantes affichent les listes précédemment définies.

```
>>> animaux
>>> tailles
>>> mixte
```

3. Que doit-on saisir pour que obtenir `"singe"` ? Vérifiez. . . . . .

Source : *Inspiré de l'université de Paris Diderot*

Il existe de plus deux manière de créer une liste vide nommée L :

- `L = []`
- `L = list()`

Et il est possible d'ajouter l'élément $elem$ directement à la fin de la liste L en saisissant :

- `L.append(elem)`

Enfin, il est possible de connaître le nombre d'éléments de la liste L en saisissant :

- `len(L)`

4. Rajouter le dodo et le bonobo à la liste des animaux puis vérifiez que la liste contient bien 6 éléments. Appelez-moi.

3) Visualiser l'avancée du caillou

1. Dans l'éditeur, définissez une nouvelle fonction `def parcours(N):` qui devra renvoyer une liste contenant les N premiers termes de la suite $d(n)$. On ne cherchera pas forcément à optimiser le code.
2. Demandez la liste des 5 premiers termes et vérifiez la cohérence avec les résultats précédemment obtenus.
3. En demandant un plus grand parcours, comment se porte notre conjecture d'une limite de 8 ?

4) Visualisation graphique – pour aller plus loin

Il est possible grâce au liste et à la bibliothèque **matplotlib** de représenter la liste L contenant les valeurs $1;2;4;-1$ et 5 sous forme d'une droite brisée.

```
1 import matplotlib.pyplot as plt
2
3 L=[1,2,4,-1,5]
4 plt.plot(L)
5 plt.show()
```

1. Modifiez votre fonction précédente (ou créez-en une nouvelle) afin de visualiser graphiquement le parcours.
2. Testez votre fonction avec différentes valeurs et consolidez ainsi notre conjecture.
3. Appelez-moi pour vous féliciter.