7 Fichiers

7.1 Lecture dans un fichier

Une grande partie de l'information en biologie est stockée sous forme de texte dans des fichiers. Pour traiter cette information, vous devez le plus souvent lire ou écrire dans un ou plusieurs fichiers. Python possède pour cela de nombreux outils qui vous simplifient la vie.

7.1.1 Méthode `.readlines()`

Avant de passer à un exemple concret, créez un fichier dans un éditeur de texte que vous enregistrerez dans votre répertoire courant avec le nom `zoo.txt` et le contenu suivant :

```
girafe
tigre
singe
souris
```

Ensuite, testez le code suivant dans l'interpréteur Python :

```
>>> filin = open("zoo.txt", "r")
>>> filin
<_io.TextIOWrapper name='zoo.txt' mode='r' encoding='UTF-8'>
>>> filin.readlines()
['girafe\n', 'tigre\n', 'singe\n', 'souris\n']
>>> filin.close()
>>> filin.readlines()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
```

Il y a plusieurs commentaires à faire sur cet exemple :

Ligne 1. L'instruction `open()` ouvre le fichier `zoo.txt`. Ce fichier est ouvert en lecture seule, comme l'indique le second argument `r` (pour *read*) de la fonction `open()`. Remarquez que le fichier n'est pas encore lu, mais simplement ouvert (*un peu comme lorsqu'on ouvre un livre, mais qu'on ne l'a pas encore lu*). Le curseur de lecture est prêt à lire le premier caractère du fichier. L'instruction `open("zoo.txt", "r")` suppose que le fichier `zoo.txt` est dans le répertoire depuis lequel l'interpréteur Python a été lancé. Si ce n'est pas le cas, il faut préciser le **chemin d'accès** au fichier. Par exemple, `/home/pierre/zoo.txt` pour Linux ou Mac OS X ou `C:\Users\pierre\zoo.txt` pour Windows.

Ligne 2. Lorsqu'on affiche le contenu de la variable `filin`, on se rend compte que Python la considère comme un objet de type fichier ouvert (ligne 3).

Ligne 4. Nous utilisons à nouveau la syntaxe `objet.méthode()` (présentée dans le chapitre 3 *Affichage*). Ici la méthode `.readlines()` agit sur l'objet `filin` en déplaçant le curseur de lecture du début à la fin du fichier, puis elle renvoie une liste contenant toutes les lignes du fichier (*dans notre analogie avec un livre, ceci correspondrait à lire toutes les lignes du livre*).

Ligne 6. Enfin, on applique la méthode `.close()` sur l'objet `filin`, ce qui, vous vous en doutez, ferme le fichier (*ceci correspondrait à fermer le livre*). Vous remarquerez que la méthode `.close()` ne renvoie rien mais modifie l'état de l'objet `filin` en fichier fermé. Ainsi, si on essaie de lire à nouveau les lignes du fichier, Python renvoie une erreur car il ne peut pas lire un fichier fermé (lignes 7 à 10).

Voici maintenant un exemple complet de lecture d'un fichier avec Python.

```
1  >>> filin = open("zoo.txt", "r")
2  >>> lignes = filin.readlines()
3  >>> lignes
4  ['girafe\n', 'tigre\n', 'singe\n', 'souris\n']
5  >>> for ligne in lignes:
6  ...     print(ligne)
7  ...
8  girafe
9
10 tigre
11
12 singe
13
14 souris
15
16 >>> filin.close()
```

Vous voyez qu'en cinq lignes de code, vous avez lu, parcouru le fichier et affiché son contenu.

Remarque

- Chaque élément de la liste `lignes` est une chaîne de caractères. C'est en effet sous forme de chaînes de caractères que Python lit le contenu d'un fichier.
- Chaque élément de la liste `lignes` se termine par le caractère `\n`. Ce caractère un peu particulier correspond au « saut de ligne » qui permet de passer d'une ligne à la suivante. Ceci est codé par un caractère spécial que l'on représente par `\n`. Vous pourrez parfois rencontrer également la notation octale `\012`. Dans la suite de cet ouvrage, nous emploierons aussi l'expression « retour à la ligne » que nous trouvons plus intuitive.
- Par défaut, l'instruction `print()` affiche quelque chose puis revient à la ligne. Ce retour à la ligne dû à `print()` se cumule alors avec celui de la fin de ligne (`\n`) de chaque ligne du fichier et donne l'impression qu'une ligne est sautée à chaque fois.

Il existe en Python le mot-clé `with` qui permet d'ouvrir et de fermer un fichier de manière efficace. Si pour une raison ou une autre l'ouverture ou la lecture du fichier conduit à une erreur, l'utilisation de `with` garantit la bonne fermeture du fichier, ce qui n'est pas le cas dans le code précédent. Voici donc le même exemple avec `with` :

```
>>> with open("zoo.txt", 'r') as filin:
...     lignes = filin.readlines()
...     for ligne in lignes:
...         print(ligne)
...
girafe

tigre

singe

souris

>>>
```

Remarque

- L'instruction `with` introduit un bloc d'indentation. C'est à l'intérieur de ce bloc que nous effectuons toutes les opérations sur le fichier.
- Une fois sorti du bloc d'indentation, Python fermera **automatiquement** le fichier. Vous n'avez donc plus besoin d'utiliser la méthode `.close()`.

7.1.2 Méthode `.read()`

Il existe d'autres méthodes que `.readlines()` pour lire (et manipuler) un fichier. Par exemple, la méthode `.read()` lit tout le contenu d'un fichier et renvoie une chaîne de caractères unique.

```
>>> with open("zoo.txt", "r") as filin:
...     filin.read()
...
'girafe\ntigre\nsinge\nsouris\n'
>>>
```

7.1.3 Méthode `.readline()`

La méthode `.readline()` (sans `s` à la fin) lit une ligne d'un fichier et la renvoie sous forme de chaîne de caractères. À chaque nouvel appel de `.readline()`, la ligne suivante est renvoyée. Associée à la boucle `while`, cette méthode permet de lire un fichier ligne par ligne.

```
>>> with open("zoo.txt", "r") as filin:
...     ligne = filin.readline()
...     while ligne != "":
...         print(ligne)
...         ligne = filin.readline()
...
girafe

tigre

singe

souris

>>>
```

7.1.4 Itérations directe sur le fichier

Python essaie de vous faciliter la vie au maximum. Voici un moyen à la fois simple et élégant de parcourir un fichier.

```
>>> with open("zoo.txt", "r") as filin:
...     for ligne in filin:
...         print(ligne)
...
girafe

tigre

singe

souris

>>>
```

L'objet `filin` est « itérable », ainsi la boucle `for` va demander à Python d'aller lire le fichier ligne par ligne.

Conseils

Privilégiez cette méthode par la suite.

Remarque

Les méthodes abordées précédemment permettent d'accéder au contenu d'un fichier, soit ligne par ligne (méthode `.readline()`), soit globalement en une seule chaîne de caractères (méthode `.read()`), soit globalement avec les lignes différenciées sous forme d'une liste de chaînes de

caractères (méthode `.readlines()`). Il est également possible en Python de se rendre à un endroit particulier d'un fichier avec la méthode `.seek()` mais qui sort du cadre de cet ouvrage.

7.2 Écriture dans un fichier

Écrire dans un fichier est aussi simple que de le lire. Voyez l'exemple suivant :

```
1  >>> animaux2 = ["poisson", "abeille", "chat"]
2  >>> with open("zoo2.txt", "w") as filout:
3  ...     for animal in animaux2:
4  ...         filout.write(animal)
5  ...
6  7
7  7
8  4
```

Quelques commentaires sur cet exemple :

Ligne 1. Création d'une liste de chaînes de caractères `animaux2`.

Ligne 2. Ouverture du fichier `zoo2.txt` en mode écriture, avec le caractère `w` pour *write*. L'instruction `with` crée un bloc d'instructions qui doit être indenté.

Ligne 3. Parcours de la liste `animaux2` avec une boucle `for`.

Ligne 4. À chaque itération de la boucle, nous avons écrit chaque élément de la liste dans le fichier. La méthode `.write()` s'applique sur l'objet `filout`. Notez qu'à chaque utilisation de la méthode `.write()`, celle-ci nous affiche le nombre d'octets (équivalent au nombre de caractères) écrits dans le fichier (lignes 6 à 8). Ceci est valable uniquement dans l'interpréteur, si vous créez un programme avec les mêmes lignes de code, ces valeurs ne s'afficheront pas à l'écran.

Si nous ouvrons le fichier `zoo2.txt` avec un éditeur de texte, voici ce que nous obtenons :

```
poissonabeillechat
```

Ce n'est pas exactement le résultat attendu car implicitement nous voulions le nom de chaque animal sur une ligne. Nous avons oublié d'ajouter le caractère fin de ligne après chaque nom d'animal.

Pour ce faire, nous pouvons utiliser l'écriture formatée :

```
1  >>> animaux2 = ["poisson", "abeille", "chat"]
2  >>> with open("zoo2.txt", "w") as filout:
3  ...     for animal in animaux2:
4  ...         filout.write(f"{animal}\n")
5  ...
6  8
```

```
8
5
```

Ligne 4. L'écriture formatée vue au chapitre 3 *Affichage* permet d'ajouter un retour à la ligne (`\n`) après le nom de chaque animal.

Lignes 6 à 8. Le nombre d'octets écrits dans le fichier est augmenté de 1 par rapport à l'exemple précédent car le caractère retour à la ligne compte pour un seul octet.

Le contenu du fichier `zoo2.txt` est alors :

```
poisson
abeille
chat
```

Vous voyez qu'il est extrêmement simple en Python de lire ou d'écrire dans un fichier.

7.3 Ouvrir deux fichiers avec l'instruction `with`

On peut avec l'instruction `with` ouvrir deux fichiers (ou plus) en même temps. Voyez l'exemple suivant :

```
with open("zoo.txt", "r") as fichier1, open("zoo2.txt", "w") as fichier2:
    for ligne in fichier1:
        fichier2.write("* " + ligne)
```

Si le fichier `zoo.txt` contient le texte suivant :

```
souris
girafe
lion
singe
```

alors le contenu de `zoo2.txt` sera :

```
* souris
* girafe
* lion
* singe
```

Dans cet exemple, `with` permet une notation très compacte en s'affranchissant de deux méthodes `.close()` .

Si vous souhaitez aller plus loin, sachez que l'instruction `with` est plus générale et est utilisable dans d'autres contextes.

7.4 Note sur les retours à la ligne sous Unix et sous Windows

Conseil : si vous êtes débutant, vous pouvez sauter cette rubrique.

On a vu plus haut que le caractère spécial `\n` correspondait à un retour à la ligne. C'est le standard sous Unix (Mac OS X et Linux).

Toutefois, Windows utilise deux caractères spéciaux pour le retour à la ligne : `\r` correspondant à un retour chariot (hérité des machines à écrire) et `\n` comme sous Unix.

Si vous avez commencé à programmer en Python 2, vous aurez peut-être remarqué que selon les versions, la lecture de fichier supprimait parfois les `\r` et d'autres fois les laissait. Heureusement, la fonction `open()` dans Python 3 gère tout ça automatiquement et renvoie uniquement des sauts de ligne sous forme d'un seul `\n` (même si le fichier a été conçu sous Windows et qu'il contient initialement des `\r`).

7.5 Importance des conversions de types avec les fichiers

Vous avez sans doute remarqué que les méthodes qui lisent un fichier (par exemple `.readlines()`) vous renvoient systématiquement des chaînes de caractères. De même, pour écrire dans un fichier il faut fournir une chaîne de caractères à la méthode `.write()`.

Pour tenir compte de ces contraintes, il faudra utiliser les fonctions de conversions de types vues au chapitre 2 *Variables* : `int()`, `float()` et `str()`. Ces fonctions de conversion sont essentielles lorsqu'on lit ou écrit des nombres dans un fichier.

En effet, les nombres dans un fichier sont considérés comme du texte, donc comme des chaînes de caractères, par la méthode `.readlines()`. Par conséquent, il faut les convertir (en entier ou en *float*) si on veut effectuer des opérations numériques avec.

7.6 Du respect des formats de données et de fichiers

Maintenant que vous savez lire et écrire des fichiers en Python, vous êtes capables de manipuler beaucoup d'information en biologie. Prenez garde cependant aux formats de fichiers, c'est-à-dire à la manière dont est stockée l'information biologique dans des fichiers. Nous vous renvoyons pour cela à l'annexe A *Quelques formats de données rencontrés en biologie*.

7.7 Exercices

Conseil : pour ces exercices, créez des scripts puis exécutez-les dans un *shell*.

7.7.1 Moyenne des notes

Le fichier `notes.txt` contient les notes obtenues par des étudiants pour le cours de Python. Chaque ligne du fichier ne contient qu'une note.

Téléchargez le fichier `notes.txt` et enregistrez-le dans votre répertoire de travail. N'hésitez pas à l'ouvrir avec un éditeur de texte pour voir à quoi il ressemble.

Créez un script Python qui lit chaque ligne de ce fichier, extrait les notes sous forme de *float* et les stocke dans une liste.

Terminez le script en calculant et affichant la moyenne des notes avec deux décimales.

7.7.2 Admis ou recalé

Le fichier `notes.txt` contient les notes obtenues par des étudiants pour le cours de Python. Chaque ligne du fichier ne contient qu'une note.

Téléchargez le fichier `notes.txt` et enregistrez-le dans votre répertoire de travail. N'hésitez pas l'ouvrir avec un éditeur de texte pour voir à quoi il ressemble.

Créez un script Python qui lit chaque ligne de ce fichier, extrait les notes sous forme de *float* et les stocke dans une liste.

Le script réécrira ensuite les notes dans le fichier `notes2.txt` avec une note par ligne suivie de « recalé » si la note est inférieure à 10 et « admis » si la note est supérieure ou égale à 10. Toutes les notes seront écrites avec une décimale. À titre d'exemple, voici les 3 premières lignes attendues pour le fichier `notes2.txt` :

```
13.5 admis
17.0 admis
9.5 recalé
```

7.7.3 Spirale (exercice +++)

Créez un script `spirale.py` qui calcule les coordonnées cartésiennes d'une spirale à deux dimensions.

Les coordonnées cartésiennes x_A et y_A d'un point A sur un cercle de rayon r s'expriment en fonction de l'angle θ représenté sur la figure 1 comme :

$$x_A = \cos(\theta) \times r$$

$$y_A = \sin(\theta) \times r$$

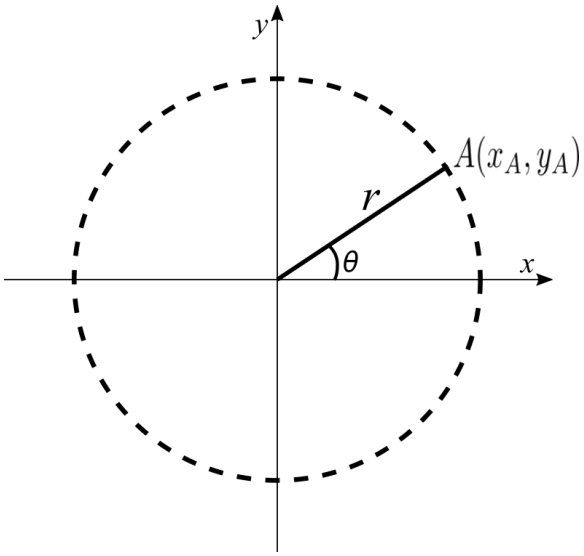

Figure 1. Point A sur le cercle de rayon r.

Pour calculer les coordonnées cartésiennes qui décrivent la spirale, vous allez faire varier deux variables en même temps :

- l'angle θ, qui va prendre des valeurs de 0 à 4π radians par pas de 0.1, ce qui correspond à deux tours complets ;
- le rayon du cercle r, qui va prendre comme valeur initiale 0.5 puis que vous allez incrémenter (c'est-à-dire augmenter) par pas de 0.1.

Les fonctions trigonométriques sinus et cosinus sont disponibles dans le module *math* que vous découvrirez plus en détails dans le chapitre 8 *Modules*. Pour les utiliser, vous ajouterez au début de votre script l'instruction :

```
import math
```

La fonction sinus sera `math.sin()` et la fonction cosinus `math.cos()`. Ces deux fonctions prennent comme argument une valeur d'angle en radian. La constante mathématique π sera également accessible grâce à ce module via `math.pi`. Par exemple :

```
>>> math.sin(0)
0.0
>>> math.sin(math.pi/2)
1.0
>>> math.cos(math.pi)
-1.0
```

Sauvegardez ensuite les coordonnées cartésiennes dans le fichier `spirale.dat` en respectant le format suivant :

- un couple de coordonnées (x_A et y_A) par ligne ;
- au moins un espace entre les deux coordonnées x_A et y_A ;

- les coordonnées affichées sur 10 caractères avec 5 chiffres après la virgule.

Les premières lignes de `spirale.dat` devrait ressembler à :

```
   0.50000    0.00000
   0.59700    0.05990
   0.68605    0.13907
   0.76427    0.23642
   0.82895    0.35048
   0.87758    0.47943
   [...]      [...]
```

Une fois que vous avez généré le fichier `spirale.dat`, visualisez votre spirale avec le code suivant (que vous pouvez recopier dans un autre script ou à la suite de votre script `spirale.py`) :

```
import matplotlib.pyplot as plt

x = []
y = []
with open("spirale.dat", "r") as f_in:
    for line in f_in:
        coords = line.split()
        x.append(float(coords[0]))
        y.append(float(coords[1]))

plt.figure(figsize=(8,8))
mini = min(x+y) * 1.2
maxi = max(x+y) * 1.2
plt.xlim(mini, maxi)
plt.ylim(mini, maxi)
plt.plot(x, y)
plt.savefig("spirale.png")
```

Visualisez l'image `spirale.png` ainsi créée.

Remarque

Le module *matplotlib* est utilisé ici pour la visualisation de la spirale. Son utilisation est détaillée dans le chapitre 17 *Quelques modules d'intérêt en bioinformatique*.

Essayez de jouer sur les paramètres θ et r, et leur pas d'incrémentation, pour construire de nouvelles spirales.