Cours 09 : Programmation orientée-objet en Python – v6

Connexion à Capytale :

- **login.capytale.fr** (choisir Connexion sans ENT, en bas colonne de droite)

Notebook pour ce cours :

- Outils Python : caaf-3184957
- Notebook-prof (pour accéder aux démos faites au tableau) : 45b0-3329984

Problématiques de sauvegarde

I) Sauvegarde dans un fichier

On s'intéresse à la sauvegarde d'une séquence ADN dans un fichier.
Une séquence ADN a la forme d'une suite de bases : adénine (A), cytosine (C), guanine (G) ou thymine (T), que l'on représentera en machine par une chaîne de caractères composée de minuscules, `'a'`, `'c'`, `'g'` ou `'t'`.
Une séquence aléatoire a été générée sur le site https://www.bioinformatics.org/sms2/random_dna.html et enregistrée dans le fichier **sequenceADN_alea.txt**, annexé au notebook.

1) Lecture dans un fichier

On rappelle la façon de lire en une fois le contenu d'un fichier texte dans un fichier :

```
fd = open(nomfichier)
contenu = fd.read()
fd.close()
```

ou, de façon équivalente :

```
with open(nomfichier) as fd:
    contenu = fd.read()
```

`fd` est un nom pour le « descripteur de fichier » créé par l'appel à `open`, et qui permet d'interagir avec le système d'exploitation pour ouvrir le fichier dont le nom est `nomfichier`. L'appel à la méthode `.close` permet de fermer l'accès au fichier en lecture et de rendre à nouveau le fichier disponible pour d'autres opérations. **On rappelle que le nom du fichier est une chaîne de caractères incluant l'extension du fichier**.

Q1. Lire le contenu du fichier et le stocker dans une variable `sequenceADN`, et déterminer le nombre de caractères lus dans le fichier.

Q2. En déduire le nombre d'octets occupés par le fichier. Vérifier grâce à la séquence d'instructions suivantes :

```
import os
os.path.getsize(nomfichier)
```

2) Un premier encodage des données

On se propose de remplacer les lettres représentant les quatre bases par un chiffre `'1'`, `'2'`, `'3'` ou `'4'`.
Pour cela, on définit le dictionnaire suivant :

```
D0 = {'a': '1', 'c': '2', 'g': '3', 't': '4'}
```

qui permet d'accéder au code d'une base de la façon suivante :

```
>>> D0['a']
'1'
```

On considère un extrait de la séquence d'ADN, constitué de ces dix premières bases :

```
extrait = contenu[0:10]
```

que l'on convertit en une liste de caractères :

```
liste_test = list(extrait)
```

de sorte que l'on a ;

```
>>> liste_test
['a', 't', 'c', 'g', 't', 'c', 'c', 'a', 'a', 'a']
```

Q3. Construire, à l'aide d'une boucle ou d'une compréhension de liste, la liste `liste_codee`, qui est une copie de la liste `liste_test` dans laquelle on a remplacé les lettres `'a'`, `'c'`, `'g'` et `'t'` par le chiffre qui les code.
On doit obtenir :

```
>>> liste_codee
['1', '4', '2', '3', '4', '2', '2', '1', '1', '1']
```

3) Enregistrement des données encodées

On souhaite enregistrer la séquence ADN complète, `sequenceADN`, dans un fichier, sous forme codée comme en **Q3**.
On rappelle la façon d'écrire des données textuelles dans un fichier, ici une chaîne de caractères nommée `chaine_a_ecrire`, que l'on écrit dans un fichier dont le nom est donné par la variable `nomfichier` :

```
fd = open(nomfichier, mode ="w")
fd.write(chaine_a_ecrire)
fd.close()
```

ou, de façon équivalente :

```
with open(nomfichier) as fd:
    fd.write(chaine_a_ecrire)
```

Pour un fichier de texte, le nom du fichier aura pour extension **.txt**.
On signale que l'on peut répéter les instructions `fd.write(...)` à volonté, tant que l'accès au fichier n'est pas fermé, afin d'écrire de nouvelles chaînes de caractères dans le fichier.
Enfin, il faut savoir que, si le fichier dont le nom est donné existe déjà, le fichier déjà existant sera écrasé par le nouveau.

Q4. Écrire dans un fichier **sequence_test_codee.txt**, les chiffres de la liste `liste_codee`. On pourra faire un appel à la méthode `.write` pour écrire chaque chiffre de la liste, où utiliser la méthode `.join`, qui permet de construire une chaîne unique à partir d'une liste de sous-chaînes, et d'un séparateur, éventuellement égal à une chaîne vide :

```
>>> ';'.join(['abc', 'de', 'fgh'])
'abc;de;fgh'
>>> ''.join(['abc', 'de', 'fgh'])
'abcdefgh'
```

afin d'écrire en une seule fois toute la chaîne codée.

Q5. Donner par le calcul la taille du fichier créé, et vérifier à l'aide d'une instruction Python. A-t-on gagné de l'espace en mémoire grâce à l'encodage choisi ? Expliquer pourquoi.

4) Un deuxième encodage des données

Q6. Combien de valeurs peut-on coder sur 2 bits ? sur 3 bits ? sur un octet ?

Q7. Combien de bits au minimum sont nécessaires pour définir un codage binaire pour les quatre bases d'une séquence ADN ? Justifier.

Q8. Choisir un codage binaire pour les quatre bases en faisant correspondre à chaque base une chaîne de caractères composée de zéros et de uns, et le définir à l'aide d'un dictionnaire, `D`, comme au **2)**. Les chaînes de caractères codant chaque base devront être toutes de même longueur.

Q9. À l'aide du codage choisi, combien de bases peut-on coder sur un octet si les zéros et les uns sont stockés sous forme de bits ? En déduire quelle serait la taille d'un fichier dans lequel on stockerait la séquence ADN complète contenue dans la variable `sequenceADN` ainsi codée par une suite de bits.

Q10. Définir, à partir du dictionnaire `D`, un dictionnaire `Dreciproque`, associant au codage de chaque base sous la forme d'une chaîne de zéros et de uns, la base que cette chaîne code.

Q11. Quelle est la suite de bases codée par l'octet `"01110001"` ?

Q12. Définir une fonction `encodage(chaine, D)`, prenant en entrée une chaîne de caractères composée des lettres « a », « c », « g » et « t », et le dictionnaire D, et renvoyant une liste d'octets correspondant à l'encodage de cette chaîne. On vérifiera à l'aide d'une assertion que la chaîne en entrée permet de coder la chaîne en entrée sur un nombre entier d'octets.

Q13. Définir une fonction `decodage(L, Dreciproque)`, prenant en entrée une liste d'octets codant une séquence ADN et renvoyant la séquence codée sous la forme d'une liste de bases.

II) Écriture d'un entier dans une base b

On rappelle que pour tout entier $b > 2$, il existe pour tout entier une décomposition unique de n sous la forme :

$$n = \sum_{k=0}^{m-1} c_k b^k = c_{m-1}b^{m-1} + \cdots + c_1 b^1 + c_0 b^0$$

où les c_i sont compris entre 0 et $b-1$ et $c_{m-1} \neq 0$.
La valeur de l'entier m donne le nombre de chiffres de l'écriture en base b de l'entier n.
La valeur des coefficients c_i donnent les chiffres de l'écriture en base b de l'entier n, que l'on peut noter

$$n = (c_{m-1} \dots c_1 c_0)_b.$$

- en base 10 ($b = 10$), les chiffres pour représenter les entiers de 0 à $b-1$ sont les chiffres usuels `'0'`, `'1'`, …, `'9'`.
  On a ainsi, par exemple, pour l'entier 24705, qui s'écrit avec 5 chiffres en base 10 :
  $$(24705)_{10} = 2 \times 10^4 + 4 \times 10^3 + 7 \times 10^2 + 0 \times 10^1 + 5 \times 10^0,$$
  ce qui correspond au fait que :
  $$24705 = 20000 + 4000 + 700 + 5.$$

- en base 2 ($b = 2$), les chiffres pour représenter les entiers de 0 à $b-1$ sont les chiffres usuels `'0'` et `'1'`.
  On a ainsi, par exemple, pour l'entier 14, qui s'écrit avec 4 chiffres en base 2 :
  $$(1110)_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0.$$
  ce qui correspond au fait que :
  $$14 = 8 + 4 + 2.$$

- en base 16 ($b = 16$), les chiffres pour représenter les entiers de 0 à 9 sont les chiffres usuels `'0'`, `'1'`, …, `'9'`, et le chiffres pour représenter les entiers de 10 à $b - 1 = 15$, sont les lettres `'a'`, `'b'`, `'c'`, `'d'` et `'f'`.
  On a ainsi, par exemple, pour l'entier 3883, qui s'écrit avec 3 chiffres en base 16 :
  $$(f2b)_2 = 15 \times 16^2 + 2 \times 16^1 + 11 \times 2^0.$$
  ce qui correspond au fait que :
  $$3883 = 15 \times 256 + 2 \times 16 + 11 \times 1.$$

1) Décodage d'une écriture en base b

Q14. Calculer l'entier n_1, dont l'écriture en base 2 est `"11010111"`. Vérifier en évaluant l'expression `int("11010111", 2)`.

Q15. Calculer l'entier n_2, dont l'écriture en base 16 est `"c4bf0"`. Vérifier en évaluant l'expression `int("c4bf0", 16)`.

Q16. Définir une fonction `bin_to_int`, prenant en argument une chaîne de caractères, `ch`, composée de zéros et de uns représentant l'écriture binaire d'un entier n et calculant et renvoyant l'entier n.

Q17. Modifier la fonction précédente en une fonction `hex_to_int`, prenant en argument une chaîne de caractères, `ch`, composée des chiffres de la base 16 et renvoyant l'entier dont `ch` est l'écriture en base 16.

2) Obtention de l'écriture d'un entier $n > 0$ en base b

On peut observer, sur la décomposition d'un entier $n > 0$ dans une base $b > 2$:

$$n = \sum_{k=0}^{m-1} c_k b^k = c_{m-1}b^{m-1} + \cdots + c_1 b^1 + c_0 b^0$$

que l'entier n peut s'écrire encore sous la forme

$$n = b\left(\sum_{k=1}^{m-1} c_k b^{k-1}\right) + c_0 = b(c_{m-1}b^{m-2} + \cdots + c_1 b^0) + c_0$$

de sorte que l'on a

$$n = (c_{m-1} \dots c_1 c_0)_b = b \times (c_{m-2} \dots c_1)_b + c_0.$$

Puisque $0 \leq c_0 < b$, on peut identifier cette écriture de n à la division euclidienne de n par b :

$$n = bq + r, \text{ où } 0 \leq r < b.$$

On en conclut, par unicité du quotient et du reste dans la division euclidienne, que :

$$c_0 = r \text{ et } q = (c_{m-2} \dots c_1)_b.$$

Il s'ensuit que l'on peut obtenir le chiffre des unités de l'écriture de n dans une base b, c_0, en effectuant la division euclidienne de n par b, (c_0 sera égal au reste de cette division), puis les autres chiffres de l'écriture de n dans la base b en itérant le procédé sur q, le quotient dans la division euclidienne de n par b.

- Par exemple, pour l'entier 311 en base 2 :

entier	division euclidienne par 2	quotient	reste	Chiffre de l'écriture en base 2
311	$311 = 2 \times 155 + 1$	155	1	$c_0 =$ "1"
155	$155 = 2 \times 77 + 1$	77	1	$c_1 =$ "1"
77	$77 = 2 \times 38 + 1$	38	1	$c_2 =$ "1"
38	$38 = 2 \times 19 + 0$	19	0	$c_3 =$ "0"
19	$19 = 2 \times 9 + 1$	9	1	$c_4 =$ "1"
9	$9 = 2 \times 4 + 1$	4	1	$c_5 =$ "1"
4	$4 = 2 \times 2 + 0$	2	0	$c_6 =$ "0"
2	$2 = 2 \times 1 + 0$	1	0	$c_7 =$ "0"
1	$1 = 2 \times 0 + 1$	0	1	$c_8 =$ "1"
0	–	–	–	–

- Algorithme en Python :

Algorithme

```
1. n = 311
2.
3. b = 2
4.
5. q = n
6. while q > 0:
7.     q, r = q // b, q % b
8.     print(q, r)
```

Résultat d'exécution :

```
q = 155 r = 1
q = 77 r = 1
q = 38 r = 1
q = 19 r = 0
q = 9 r = 1
q = 4 r = 1
q = 2 r = 0
q = 1 r = 0
q = 0 r = 1
```

Conclusion :

$$311 = (100110111)_2.$$

- Vérification du résultat par appel à une fonction prédéfinie :

```
>>> bin(311)
'0b100110111'
```

- Par exemple, pour l'entier 438 en base 16 :

entier	division euclidienne par 2	quotient	reste	Chiffre de l'écriture en base 2
43890	$43890 = 16 \times 2743 + 2$	155	2	$c_0 =$ "2"
2743	$2743 = 16 \times 171 + 7$	171	7	$c_1 =$ "7"
171	$171 = 16 \times 10 + 11$	10	11	$c_2 =$ "b"
10	$11 = 16 \times 0 + 10$	0	10	$c_3 =$ "a"
0	–	–	–	–

- Algorithme en Python :

Algorithme

```
1. n, b = 43890, 16
2. q = n
3. while q > 0:
4.     q, r = q // b, q % b
5.     print(q, r)
```

Résultat d'exécution :

```
>>>
q = 2743 r = 2
q = 171 r = 7
q = 10 r = 11
q = 0 r = 10
```

Conclusion :

$$4389 = (ab72)_{16}.$$

- Vérification du résultat par appel à une fonction prédéfinie :

```
>>> hex(43890)
'0xab72'
```

Q18. Déterminer l'écriture en base 2 des entiers 18, 32 et 115.

Q19. Déterminer le plus petit entier dont l'écriture binaire comporte trois chiffres, et le plus grand entier dont l'écriture binaire comporte quatre chiffres.

Q20. Déterminer l'écriture en base 16, des entiers 30, 512 et 14707.

Q21. Quel est le plus grand entier dont l'écriture en base 16 comporte trois chiffres ? Quel est le plus petit entier dont l'écriture en base 16 comporte quatre chiffres ?

III) Codes préfixes

En informatique :

- on appelle **préfixe de longueur p** d'une chaîne de caractères `ch` de longueur m, toute sous-chaîne obtenue en ne gardant de la chaîne `ch` que ses p premiers caractères (on supposera $p \leq m$).
  Ainsi, avec les notations du *slicing*, pour tout $p \in [\![0\,;m]\!]$, le préfixe de longueur p d'une chaîne `ch` de longueur m sera donné par `ch[0:p]`, qui s'écrit encore, en notation abrégée, `ch[:p]`.
- on appelle **codage préfixe**, ou ***code préfixe***, un codage pour lequel un code n'est un préfixe d'aucun autre code.

Par exemple, le codage des caractères `'a'`, `'b'`, `'c'`, `'d'`, `'e'`, `'f'` et `'g'` défini par le dictionnaire

```
D1 = {'a': '01101', 'b': '1101', 'c': '011', 'd': '10110',
      'e': '01110', 'f': '01', 'g': '101'}
```

n'est pas un codage préfixe, car le code `'011'` (qui code la lettre `'c'`) est un préfixe du code de la lettre `'a'`.

Q22. Citer d'autres codes du dictionnaire `D1` qui sont un préfixe d'un autre code de `D1`.

Q23. Quel est le codage du mot `"gaffe"` ?

Q24. Est-il possible de retrouver le mot codé par la chaîne `'01101101101011110'` ?

On considère maintenant le codage des caractères `'a'`, `'b'`, `'c'`, `'d'`, `'e'`, `'f'` et `'g'` défini par le dictionnaire

```
D2 = {'a': '01101', 'b': '1101', 'c': '1111', 'd': '10110',
      'e': '01110', 'f': '00', 'g': '100'}
```

Q25. Écrire un script Python qui permette de vérifier que le codage défini par le dictionnaire `D2` est préfixe.

Q26. Décoder le texte suivant `"11010110111011010001110"`.

Q27. Écrire une fonction Python `decode(ch)`, permettant de décoder une chaîne `ch` encodée par le codage défini par le dictionnaire `D2`. On construira pour cela le dictionnaire, `D2inv`, « réciproque du dictionnaire `D2`, défini par `D2inv = {val : cle for cle, val in D2.items()}`

Q28. Tester la fonction avec la chaîne `"11010110111011010001110"`.