8 Modules

8.1 Définition

Les modules sont des programmes Python qui contiennent des fonctions que l'on est amené à réutiliser souvent (on les appelle aussi bibliothèques ou *libraries*). Les développeurs de Python ont mis au point de nombreux modules qui effectuent une quantité phénoménale de tâches. Pour cette raison, prenez le réflexe de vérifier si une partie du code que vous souhaitez écrire n'existe pas déjà sous forme de module. La plupart de ces modules sont déjà installés dans les versions standards de Python. Vous pouvez accéder à une documentation exhaustive sur le site de Python. Surfez un peu sur ce site, la quantité de modules est impressionante.

8.2 Importation de modules

Jusqu'à maintenant, nous avons rencontré une fois cette notion de module lorsque nous avons voulu tirer un nombre aléatoire.

```
>>> import random
>>> random.randint(0,10)
4
```

Regardons de plus près cet exemple :

- L'instruction `import` permet d'accéder à toutes les fonctions du module random
- Ensuite, nous utilisons la fonction (ou méthode) randint(a,b) du module `random`. Attention cette fonction renvoie un nombre entier aléatoirement entre `a` inclus et `b` inclus (contrairement à `range()` par exemple). Remarquez la notation objet `random.randint()` où la fonction `randint()` peut être considérée comme une méthode de l'objet `random`.

Il existe un autre moyen d'importer une ou des fonctions d'un module :

```
>>> from random import randint
>>> randint(0,10)
7
```

À l'aide du mot-clé `from`, vous pouvez importer une fonction spécifique d'un module donné. Remarquez que dans ce cas il est inutile de répéter le nom du module, seul le nom de ladite fonction est requis.

On peut également importer toutes les fonctions d'un module :

```
>>> from random import *
>>> x = [1, 2, 3, 4]
>>> shuffle(x)
>>> x
[2, 3, 1, 4]
>>> shuffle(x)
>>> x
[4, 2, 1, 3]
>>> randint(0,50)
46
>>> uniform(0,2.5)
0.64943174760727951
```

Comme vous l'avez deviné, l'instruction `from random import *` importe toutes les fonctions du module `random`. On peut ainsi utiliser toutes ses fonctions directement, comme par exemple `shuffle()`, qui permute une liste aléatoirement.

Dans la pratique, plutôt que de charger toutes les fonctions d'un module en une seule fois (`from random import *`), nous vous conseillons de charger le module (`import random`) puis d'appeler explicitement les fonctions voulues (`random.randint(0,2)`).

Enfin, si vous voulez vider de la mémoire un module déjà chargé, vous pouvez utiliser l'instruction `del` :

```
>>> import random
>>> random.randint(0,10)
2
>>> del random
>>> random.randint(0,10)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'random' is not defined
```

Vous pouvez constater qu'un rappel d'une fonction du module `random` après l'avoir vidé de la mémoire retourne un message d'erreur.

Enfin, il est également possible de définir un alias (un nom plus court) pour un module :

```
>>> import random as rand
>>> rand.randint(1, 10)
6
```

Dans cet exemple, les fonctions du module `random` sont accessibles via l'alias `rand`.

8.3 Obtenir de l'aide sur les modules importés

Pour obtenir de l'aide sur un module rien de plus simple, il suffit d'invoquer la commande `help()` :

```
>>> import random
>>> help(random)
...
```

On peut se promener dans l'aide avec les flèches ou les touches `page-up` et `page-down` (comme dans les commandes Unix `man`, `more` ou `less`). Il est aussi possible d'invoquer de l'aide sur une fonction particulière d'un module de la manière suivante `help(random.randint)`.

La commande `help()` est en fait une commande plus générale permettant d'avoir de l'aide sur n'importe quel objet chargé en mémoire.

```
>>> x = range(2)
>>> help(x)
Help on list object:

class list(object)
 |  list() -> new list
 |  list(sequence) -> new list initialized from sequence's items
 |
 |  Methods defined here:
 |
 |  __add__(...)
```

```
 |      x.__add__(y) <==> x+y
 |
...
```

Enfin, si on veut connaître en seul coup d'oeil toutes les méthodes ou variables associées à un objet, on peut utiliser la commande `dir` :

```
>>> import random
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'System
'TWOPI', 'WichmannHill', '_BuiltinMethodType', '_MethodType', '__all__', '__buil
'__doc__', '__file__', '__name__', '_acos', '_ceil', '_cos', '_e', '_exp', '_hex
'_inst', '_log', '_pi', '_random', '_sin', '_sqrt', '_test', '_test_generator',
'_urandom', '_warn', 'betavariate', 'choice', 'expovariate', 'gammavariate', 'ga
'getrandbits', 'getstate', 'jumpahead', 'lognormvariate', 'normalvariate',
'paretovariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate',
'shuffle', 'uniform', 'vonmisesvariate', 'weibullvariate']
>>>
```

8.4 Modules courants

Il existe une série de modules que vous serez probablement amenés à utiliser si vous programmez en Python. En voici une liste non exhaustive. Pour la liste complète, reportez-vous à la page des modules sur le site de Python :

- math : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- sys : passage d'arguments, gestion de l'entrée/sortie standard...
- os : dialogue avec le système d'exploitation (e.g. permet de sortir de Python, lancer une commande en *shell*, puis de revenir à Python).
- random : génération de nombres aléatoires.
- time : permet d'accéder à l'heure de l'ordinateur et aux fonctions gérant le temps.
- calendar : fonctions de calendrier.
- profile : permet d'évaluer le temps d'exécution de chaque fonction dans un programme (*profiling* en anglais).
- urllib2 : permet de récupérer des données sur internet depuis python.
- Tkinter : interface python avec Tk (permet de créer des objets graphiques ; nécessite d'installer Tk).
- re : gestion des expressions régulières.
- pickle : écriture et lecture de structures Python (comme les dictionnaires par exemple).

Nous vous conseillons vivement d'aller surfer sur les pages de ces modules pour découvrir toutes leurs potentialités.

Vous verrez plus tard comment créer votre propres modules lorsque vous êtes amenés à réutiliser souvent vos propres fonctions.

Enfin, notez qu'il existe de nombreux autres modules qui ne sont pas installés de base dans Python mais qui sont de grand intérêt en bioinformatique (au sens large). Citons-en quelques-uns : `numpy` (algèbre linéaire, transformée de Fourier), `biopython` (recherche dans les banques de données biologiques), `rpy` (dialogue R/Python)...

8.5 Module sys : passage d'arguments

Le module sys contient (comme son nom l'indique) des fonctions et des variables spécifiques au système, ou plus exactement à l'interpréteur lui-même. Par exemple, il permet de gérer l'entrée (*stdin*) et la sortie standard (*stdout*). Ce module est particulièrement intéressant pour récupérer les arguments passés à un script Python lorsque celui-ci est appelé en ligne

de commande. Dans cet exemple, oublions l'interpréteur et écrivons le script suivant que l'on enregistrera sous le nom `test.py` (n'oubliez pas de le rendre exécutable) :

```
#!/usr/bin/env python

import sys
print sys.argv
```

Ensuite lancez `test.py` suivi de plusieurs arguments. Par exemple :

```
poulain@cumin> python test.py salut girafe 42
['test.py', 'salut', 'girafe', '42']
```

Dans l'exemple précédent, `poulain@cumin>` représente l'invite du *shell*, `test.py` est le nom du script Python, `salut`, `girafe` et `42` sont les arguments passés au script.

La variable `sys.argv` est une liste qui représente tous les arguments de la ligne de commande, y compris le nom du script lui même qu'on peut retrouver dans `sys.argv[0]`. On peut donc accéder à chacun de ces arguments avec `sys.argv[1]`, `sys.argv[2]`...

On peut aussi utiliser la fonction `sys.exit()` pour quitter un script Python. On peut donner comme argument un objet (en général une chaîne de caractères) qui sera renvoyé au moment ou Python quittera le script. Par exemple, si vous attendez au moins un argument en ligne de commande, vous pouvez renvoyer un message pour indiquer à l'utilisateur ce que le script attend comme argument :

```
#!/usr/bin/env python

import sys

if len(sys.argv) != 2:
    sys.exit("ERREUR : il faut exactement un argument.")

#
# suite du script
#
```

Puis on l'exécute sans argument :

```
poulain@cumin> python test.py
ERREUR : il faut exactement un argument.
```

Notez qu'ici on vérifie que le script possède deux arguments car le nom du script lui-même est le premier argument et `file.txt` constitue le second.

8.6 Module os

Le module `os` gère l'interface avec le système d'exploitation.

Une fonction pratique de ce module permet de gérer la présence d'un fichier sur le disque.

```
>>> import sys
>>> import os
>>> if os.path.exists("toto.pdb"):
...     print "le fichier est présent"
... else:
...     sys.exit("le fichier est absent")
...
le fichier est absent
```

Dans cet exemple, si le fichier n'est pas présent sur le disque, on quitte le programme avec la fonction `exit()` du module `sys`.

La fonction `system()` permet d'appeler n'importe quelle commande externe.

```
>>> import os
>>> os.system("ls -al")
total 5416
drwxr-xr-x 2 poulain dsimb    4096 2010-07-21 14:33 .
drwxr-xr-x 6 poulain dsimb    4096 2010-07-21 14:26 ..
-rw-r--r-- 1 poulain dsimb  124335 2010-07-21 14:31 1BTA.pdb
-rw-r--r-- 1 poulain dsimb 4706057 2010-07-21 14:31 NC_000913.fna
-rw-r--r-- 1 poulain dsimb  233585 2010-07-21 14:30 NC_001133.fna
-rw-r--r-- 1 poulain dsimb  463559 2010-07-21 14:33 NC_001133.gbk
0
```

La commande externe `ls -al` est introduite comme une chaîne de caractères à la fonction `system()`.

8.7 Exercices

Conseil : pour les trois premiers exercices, utilisez l'interpréteur Python. Pour les exercices suivants, écrivez des scripts dans des fichiers, puis exécutez-les dans un *shell*.

1. Affichez sur la même ligne les nombres de 10 à 20 (inclus) ainsi que leur racine carrée avec 3 décimales (module math). Exemple :

   ```
   10 3.162
   11 3.317
   12 3.464
   13 3.606
   ```

2. Calculez le cosinus de pi/2 (module math).

3. Affichez la liste des fichiers du répertoire courant avec le module `os`. N'utilisez pas la fonction `os.system()` mais la fonction `os.listdir()` (lisez la documentation pour comprendre comment l'utiliser).

4. Écrivez les nombres de 1 à 10 avec 1 seconde d'intervalle (module time).

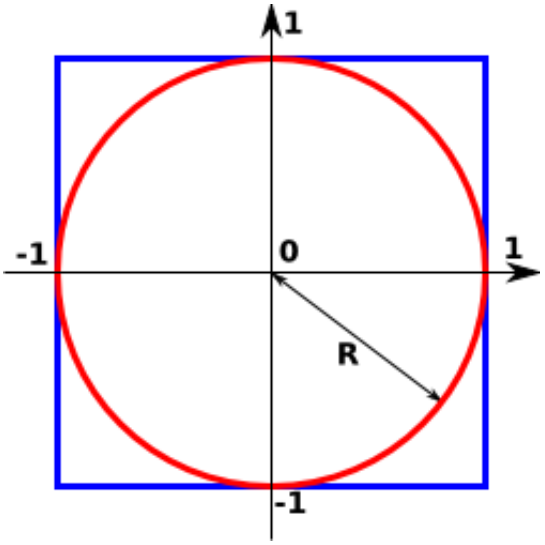
5. Générez une séquence aléatoire de 20 chiffres, ceux-ci étant des entiers tirés entre 1 et 4 (module random).

6. Générez une séquence aléatoire de 20 bases de deux manières différentes (module random).

7. Déterminez votre jour (lundi, mardi...) de naissance (module calendar).

8. Exercice +++. **Évaluation du nombre π par la méthode Monte Carlo**.

   Soit un cercle de rayon 1 (en rouge) inscrit dans un carré de coté 2 (en bleu).



L'aire du carré vaut $(2R)^2$ soit 4. L'aire du cercle vaut πR^2 soit π.

En choisissant N points aléatoires (à l'aide d'une distribution uniforme) à l'intérieur du carré, la probabilité que ces points se trouvent aussi dans le cercle est

$$p = \frac{\text{aire du cercle}}{\text{aire du carré}} = \frac{\pi}{4}$$

Soit n, le nombre points effectivement dans le cercle, il vient alors

$$p = \frac{n}{N} = \frac{\pi}{4},$$

d'où

$$\pi = 4 \times \frac{n}{N}.$$

Déterminez une approximation de π par cette méthode. Pour cela, vous allez, pour N itérations, choisir aléatoirement les coordonnées d'un point entre -1 et 1 (fonction `uniform()` du module `random`), calculer la distance entre le centre du cercle et ce point et déterminer si cette distance est inférieure au rayon du cercle. Le cas échéant, le compteur `n` sera incrémenté.

Que vaut l'approximation de π pour 100 itérations ? 500 ? 1000 ?

Conseil : pour les premiers exercices, utilisez l'interpréteur Python.

9 Plus sur les chaînes de caractères

9.1 Préambule

Nous avons déjà abordé les chaînes de caractères dans le chapitre *variables et écriture*. Ici nous allons un peu plus loin notamment avec les méthodes associées aux chaînes de caractères. Notez qu'il existe un module `string` mais qui est maintenant considéré comme obsolète depuis la version 2.5 de Python.

9.2 Chaînes de caractères et listes

Les chaînes de caractères peuvent être considérées comme des listes.

```
>>> animaux = "girafe tigre"
>>> animaux
'girafe tigre'
>>> len(animaux)
12
>>> animaux[3]
'a'
```

Nous pouvons donc utiliser certaines propriétés des listes comme les tranches :

```
>>> animaux = "girafe tigre"
>>> animaux[0:4]
'gira'
>>> animaux[9:]
'gre'
>>> animaux[:-2]
'girafe tig'
```

A contrario des listes, les chaînes de caractères présentent toutefois une différence notable, ce sont **des listes non modifiables**. Une fois définie, vous ne pouvez plus modifier un de ses éléments. Le cas échéant, Python renvoie un message d'erreur :

```
>>> animaux = "girafe tigre"
>>> animaux[4]
'f'
>>> animaux[4] = "F"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Par conséquent, si vous voulez modifier une chaîne, vous êtes obligés d'en construire une nouvelle. Pour cela, n'oubliez pas que les opérateurs de concaténation (`+`) et de duplication (`*`) (cf chapitre *variables et écriture*) peuvent vous aider. Vous pouvez également générer une liste, qui elle est modifiable, puis revenir à une chaîne.

9.3 Caractères spéciaux

Il existe certains caractères spéciaux comme le `\n` que nous avons déjà vu (pour le retour à la ligne). Le `\t` vous permet d'écrire une tabulation. Si vous voulez écrire un guillemet simple ou double (et que celui-ci ne soit pas confondus avec les guillemets de déclaration de la chaîne de caractères), vous pouvez utiliser `\'` ou `\"` ou utiliser respectivement des guillements doubles ou simple pour déclarer votre chaîne de caractères.

```
>>> print "Un retour a la ligne\npuis une tabulation\t, puis un guillemet\""
Un retour a la ligne
puis une tabulation     , puis un guillemet"
>>> print 'J\'imprime un guillemet simple'
J'imprime un guillemet simple
>>> print "Un brin d'ADN"
Un brin d'ADN
>>> print 'Python est un "super" langage'
Python est un "super" langage
```

Lorsqu'on souhaite écrire un texte sur plusieurs lignes, il est très commode d'utiliser les guillemets triples permettant de conserver le formatage (notamment les retours à la ligne) :

```
>>> x = '''souris
... chat
... abeille'''
>>> x
'souris\nchat\nabeille'
>>> print x
souris
chat
abeille
```

9.4 Méthodes associées aux chaînes de caractères

Voici quelques méthodes spécifiques aux objets de type `string` :

```
>>> x = "girafe"
>>> x.upper()
'GIRAFE'
>>> x
'girafe'
>>> 'TIGRE'.lower()
'tigre'
```

Les fonctions `lower()` et `upper()` passent un texte en minuscule et en majuscule respectivement. On remarque que l'utilisation de ces fonctions n'altèrent pas la chaîne de départ mais renvoie la chaîne transformée.

Pour mettre en majuscule la première lettre seulement, vous pouvez faire :

```
>>> x[0].upper() + x[1:]
'Girafe'
```

ou encore plus simple avec la fonction Python adéquate :

```
>>> x.capitalize()
'Girafe'
```

Il existe une méthode associée aux chaînes de caractères qui est particulièrement pratique, la fonction `split()` :

```
>>> animaux = "girafe tigre singe"
>>> animaux.split()
['girafe', 'tigre', 'singe']
```

```
>>> for animal in animaux.split():
...     print animal
...
girafe
tigre
singe
```

La fonction `split()` découpe la ligne en champs, en utilisant comme séparateur les espaces ou les tabulations. Il est possible de modifier le séparateur de champs, par exemple :

```
>>> animaux = "girafe:tigre:singe"
>>> animaux.split(":")
['girafe', 'tigre', 'singe']
```

La fonction `find()` recherche une chaîne de caractères passée en argument.

```
>>> animal = "girafe"
>>> animal.find('i')
1
>>> animal.find('afe')
3
>>> animal.find('tig')
-1
```

Si l'élément recherché est trouvé, alors l'indice du début de l'élément dans la chaîne de caractères est renvoyé. Si l'élément n'est pas trouvé, alors la valeur -1 est renvoyée.

Si l'élément recherché est trouvé plusieurs fois, seul l'indice de la première occurence est retourné :

```
>>> animaux = "girafe tigre"
>>> animaux.find("i")
1
```

On trouve aussi la fonction `replace()`, qui serait l'équivalent de la fonction de substitution de la commande Unix **sed** :

```
>>> animaux = "girafe tigre"
>>> animaux.replace("tigre", "singe")
'girafe singe'
>>> animaux.replace("i", "o")
'gorafe togre'
```

Enfin, la fonction `count()` compte le nombre d'occurences d'une chaîne de caractères passée en argument :

```
>>> animaux = "girafe tigre"
>>> animaux.count("i")
2
>>> animaux.count("z")
0
>>> animaux.count("tigre")
1
```

9.5 Conversion de types

Dans tout langage de programmation, on est souvent amené à convertir les types, c'est-à-dire passer d'un type numérique à une chaîne de caractères ou vice-versa. En Python, rien de plus simple avec les fonctions `int()`, `float()` et `str()`. Pour vous en convaincre, regardez ces exemples :

```
>>> i = 3
>>> str(i)
'3'
>>> i = '456'
>>> int(i)
456
>>> float(i)
456.0
>>> i = '3.1416'
>>> float(i)
3.1415999999999999
```

Ces conversions sont essentielles lorsqu'on lit ou écrit des nombres dans un fichier. En effet, les nombres dans un fichier sont considérés comme du texte par la fonction `readlines()`, par conséquent il faut les convertir si on veut effectuer des opérations numériques dessus.

9.6 Conversion d'une liste de chaînes de caractères en une chaîne de caractères

La conversion d'une liste de chaînes de caractères en une chaîne de caractères est un peu particulière puisqu'elle fait appelle à la fonction `join()`.

```
>>> seq = ["A", "T", "G", "A", "T"]
>>> seq
['A', 'T', 'G', 'A', 'T']
>>> "-".join(seq)
'A-T-G-A-T'
>>> " ".join(seq)
'A T G A T'
>>> "".join(seq)
'ATGAT'
```

Les éléments de la liste initiale sont concaténés les uns à la suite des autres et intercalés par un séparateur qui peut être n'importe quelle chaîne de caractères (ici, nous avons utilisé un tiret, un espace et rien).

Attention, la fonction `join()` ne s'applique qu'à une liste de chaînes de caractères.

```
>>> maliste = ["A", 5, "G"]
>>> " ".join(maliste)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sequence item 1: expected string, int found
```

Nous espérons qu'après ce petit tour d'horizon vous serez convaincu de la richesse des méthodes associées aux chaînes de caractères. Pour avoir une liste exhaustive de l'ensemble des méthodes associées à une variable particulière, vous pouvez utiliser la commande `dir()`.

```
>>> dir(animaux)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__
'__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__',
'__hash__', '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__str__', 'capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'isl
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition'
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split'
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

Pour l'instant vous pouvez ignorer les méthodes qui commencent et qui se terminent par deux tirets bas (*underscores*) « __ ».

Vous pouvez ensuite accéder à l'aide et à la documentation d'une fonction particulière avec `help()` :

```
>>> help(animaux.split)
Help on built-in function split:

split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator.
(END)
```

9.7 Exercices

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

1. Soit la liste `['girafe', 'tigre', 'singe', 'souris']`. Avec une boucle, affichez chaque élément ainsi que sa taille (nombre de caractères).
2. Soit la séquence nucléique `ATATACGGATCGGCTGTTGCCTGCGTAGTAGCGT`. Calculez la fréquence de chaque base dans cette séquence.
3. Soit la séquence protéique `ALA GLY GLU ARG TRP TYR SER GLY ALA TRP`. Transformez cette séquence en une chaîne de caractères en utilisant le code une lettre pour les acides aminés.
4. **Distance de Hamming**.
   La distance de Hamming mesure la différence entre deux séquences de même taille en sommant le nombre de positions qui, pour chaque séquence, ne correspondent pas au même acide aminé. Calculez la distance de Hamming pour les séquences `AGWPSGGASAGLAIL` et `IGWPSAGASAGLWIL`.
5. **Palindrome**.
   Un palindrome est un mot ou une phrase dont l'ordre des lettres reste le même si on le lit de gauche à droite ou de droite à gauche. Par exemple, « ressasser » et « Engage le jeu que je le gagne » sont des palindromes.
   Écrivez un script qui détermine si une chaîne de caractères est un palindrome. Pensez à vous débarasser des majuscules et des espaces. Testez si les expressions suivantes sont des palindromes : « Radar », « *Never odd or even* », « Karine alla en Iran », « Un roc si biscornu ».

6. **Mot composable**.

   Un mot est composable à partir d'une séquence de lettres si la séquence contient toutes les lettres du mot. Comme au Scrabble, chaque lettre de la séquence ne peut être utilisée qu'une seule fois. Par exemple, *coucou* est composable à partir de *uocuoceokzefhu*.

   Écrivez un script qui permet de savoir si un mot est composable à partir d'une séquence de lettres. Testez le avec différents mots et séquences.

   Remarque : dans votre script, le mot et la séquence de lettres seront des chaînes de caractères.

7. **Alphabet et pangramme**.

   Les codes ASCII des lettres minuscules de l'alphabet vont de 97 (lettre a) à 122 (lettre z). La fonction `chr()` prend en argument un code ASCII sous forme d'une entier et renvoie le caractère correspondant. Ainsi `chr(97)` renvoie `'a'`, `chr(98)` renvoie `'b'` et ainsi de suite.

   Écrivez un script qui construit une chaîne de caractères contenant toutes les lettres de l'alphabet.

   Un pangramme est une phrase comportant au moins une fois chaque lettre de l'alphabet. Par exemple, « Portez ce vieux whisky au juge blond qui fume » est un pangramme.

   Modifiez le script précédent pour déterminer si une chaîne de caractères est un pangramme ou non. Pensez à vous débarasser des majuscules le cas échéant. Testez si les expressions suivantes sont des pangrammes : « Monsieur Jack vous dactylographiez bien mieux que votre ami Wolf », « Buvez de ce whisky que le patron juge fameux ».

8. Téléchargez le fichier pdb 1BTA dans votre répertoire. Faites un script qui récupère seulement les carbones alpha et qui les affiche à l'écran.

9. En vous basant sur le script précédent, affichez à l'écran les carbones alpha des deux premiers résidus. Toujours avec le même script, calculez la distance inter atomique entre ces deux atomes.

10. En vous basant sur le script précédent, calculez les distances entre carbones alpha consécutifs. Affichez ces distances sous la forme

    ```
    numero_calpha_1 numero_calpha_2 distance
    ```

    À la fin du script, faites également afficher la moyenne des distances. La distance inter carbone alpha dans les protéines est très stable et de l'ordre de 3,8 angströms. Observez avec attention les valeurs que vous avez obtenues. Expliquez l'anomalie détectée.