université PARIS-SACLAY

M1 BIBS

Langages de scripts : Python

Séance 3 et 4 - Parcourir et Parser un fichier (PDB en l'occurrence)

(Correction)

1 Objectifs de la séance :

— Renforcer sa maîtrise des outils Python vus dans les séances précédentes
— Savoir lire un fichier
— S'initier au parsing
— Savoir construire et manipuler des structures complexes du Python

2 Initiation à la manipulation de fichier

1. La fonction open : La première chose à faire pour pouvoir lire dans un fichier, c'est de l'ouvrir dans votre programme. Pour cela, il suffit de faire :

```
fichier = open("cheminAbsoluOuRelatif", 'r')
```

   Le 'r' signifie que vous ouvrez votre fichier en lecture seule. Si vous voulez écrire dessus, il faudra mettre un 'a' ou un 'w' selon la façon dont vous voulez intervenir.

2. Parcourir un fichier ouvert : La manière classique de lire un fichier pour en extraire du contenu, est de le parcourir ligne par ligne. Pour cela, on peut récupérer une liste ordonnée contenant toutes les lignes (i.e. des chaînes de caractères) de notre fichier en faisant :

```
l_lignes = fichier.readlines()
```

   Il ne reste qu'à parcourir les lignes du fichier avec une simple boucle :

```
for ligne in l_lignes:
        #bloc d'instructions
```

3. Remarque : La méthode readlines() charge l'intégralité du fichier dans une variable. Notez que si vous manipulez plusieurs fichiers de taille importante en même temps, cela vous posera peut-être des problèmes... Dans ce cas, on peut aussi simplement accéder aux lignes une par une :

```
for ligne in fichier:
        #bloc d'instructions
```

4. Fermer un fichier : Si votre programme souhaite de nouveau utiliser ce fichier, il ne pourra pas forcément y accéder, puisqu'il a déjà été ouvert. Pensez donc à toujours le fermer en faisant :

```
fichier.close()
```

   Si vous utilisez readlines, vous pouvez fermer votre fichier juste après. Dans tous les cas, prenez l'habitude d'écrire tout de suite un `close` dès que vous écrivez un `open`.

5. Remarque : Il y a bien plus de possibilités de manipulation de fichier, mais nous n'irons pas plus loin pour l'instant, et nous n'aborderons pas non plus toutes les spécificités sur ce sujet dans cette UE. Sachez aller chercher par vous-même d'autres informations sur internet si vous en avez besoin.

3 Parser PDB

1. Qu'est-ce qu'un fichier PDB : Les fichiers PDB contiennent les coordonnées cartésiennes des atomes qui constituent la molécule, c'est-à-dire les informations qui vont permettre de visualiser et manipuler les molécules. Notez qu'il existe plusieurs formats de fichier PDB. Nous utiliserons dans cette UE le format ATOM. Voici un exemple de fichier PDB au format ATOM :

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ATOM      32   N   AARG A   -3       11.281   86.699   94.383   0.50 35.88           N
ATOM      33   N   BARG A   -3       11.296   86.721   94.521   0.50 35.60           N
ATOM      34   CA  AARG A   -3       12.353   85.696   94.456   0.50 36.67           C
ATOM      35   CA  BARG A   -3       12.333   85.862   95.041   0.50 36.42           C
ATOM      36   C   AARG A   -3       13.559   86.257   95.222   0.50 37.37           C
ATOM      37   C   BARG A   -3       12.759   86.530   96.365   0.50 36.39           C
ATOM      38   O   AARG A   -3       13.753   87.471   95.270   0.50 37.74           O
ATOM      39   O   BARG A   -3       12.924   87.757   96.420   0.50 37.26           O
ATOM      40   CB  AARG A   -3       12.774   85.306   93.039   0.50 37.25           C
ATOM      41   CB  BARG A   -3       13.428   85.746   93.980   0.50 36.60           C
ATOM      42   CG  AARG A   -3       11.754   84.432   92.321   0.50 38.44           C
ATOM      43   CG  BARG A   -3       12.866   85.172   92.651   0.50 37.31           C
ATOM      44   CD  AARG A   -3       11.698   84.678   90.815   0.50 38.51           C
ATOM      45   CD  BARG A   -3       13.374   85.886   91.406   0.50 37.66           C
ATOM      46   NE  AARG A   -3       12.984   84.447   90.163   0.50 39.94           N
ATOM      47   NE  BARG A   -3       12.644   85.487   90.195   0.50 38.24           N
ATOM      48   CZ  AARG A   -3       13.202   84.534   88.850   0.50 40.03           C
ATOM      49   CZ  BARG A   -3       13.114   85.582   88.947   0.50 39.55           C
ATOM      50   NH1AARG A   -3       12.218   84.840   88.007   0.50 40.76           N
ATOM      51   NH1BARG A   -3       14.338   86.056   88.706   0.50 40.23           N
ATOM      52   NH2AARG A   -3       14.421   84.308   88.373   0.50 40.45           N
```

2. Organisation d'un fichier PDB : Notez que les champs du fichier sont organisés en colonne et qu'il n'y a pas toujours d'espace pour séparer 2 valeurs consécutives d'une même ligne. NB : On n'utilisera donc pas la fonction split() dans cette séance. Voici à quoi correspond une partie des champs d'un fichier :

```
COLUMNS        DATA  TYPE     FIELD         DEFINITION
-------------------------------------------------------------------------------------

 1 -  6        Record name    "ATOM  "
 7 - 11        Integer        serial        Atom  serial number.
13 - 16        Atom           name          Atom name.
17             Character      altLoc        Alternate location indicator.
18 - 20        Residue name   resName       Residue name.
22             Character      chainID       Chain identifier.
23 - 26        Integer        resSeq        Residue sequence number.
27             AChar          iCode         Code for insertion of residues.
31 - 38        Real(8.3)      x             Orthogonal coordinates for X in Angstroms.
39 - 46        Real(8.3)      y             Orthogonal coordinates for Y in Angstroms.
47 - 54        Real(8.3)      z             Orthogonal coordinates for Z in Angstroms.
```

3. L'objectif de cette séance est de développer une fonction qui va pouvoir lire un fichier PDB au format ATOM et récupérer les informations qui nous intéressent dans une variable Python. Variable qui sera renvoyée par la fonction. L'idée générale est de décomposer une molécule en chaînes polypeptidiques, puis chaque chaîne en résidus, et enfin, chaque résidu en atomes, dont on récupèrera principalement les coordonnées cartésiennes. Voici la structure de la variable en question pour l'exemple précédent de l'arginine :

```
{
'A':
        {
        'reslist': ['-3'],
        '-3':
                {
                'C': {'y': 86.53, 'x': 12.759, 'z': 96.365, 'id': '37'},
                'CB': {'y': 85.746, 'x': 13.428, 'z': 93.98, 'id': '41'},
                'CA': {'y': 85.862, 'x': 12.333, 'z': 95.041, 'id': '35'},
                'CG': {'y': 85.172, 'x': 12.866, 'z': 92.651, 'id': '43'},
                'NE': {'y': 85.487, 'x': 12.644, 'z': 90.195, 'id': '47'},
                'O': {'y': 87.757, 'x': 12.924, 'z': 96.42, 'id': '39'},
                'N': {'y': 86.721, 'x': 11.296, 'z': 94.521, 'id': '33'},
                'CZ': {'y': 85.582, 'x': 13.114, 'z': 88.947, 'id': '49'},
```

```
                'CD': {'y': 85.886, 'x': 13.374, 'z': 91.406, 'id': '45'},
                'NH1': {'y': 86.056, 'x': 14.338, 'z': 88.706, 'id': '51'},
                'NH2': {'y': 84.308, 'x': 14.421, 'z': 88.373, 'id': '52'},
                'resname': 'ARG',
                'atomlist': ['N', 'N', 'CA', 'CA', 'C', 'C', 'O', 'O', 'CB', 'CB', '
                    CG', 'CG', 'CD', 'CD', 'NE', 'NE', 'CZ', 'CZ', 'NH1', 'NH1', '
                    NH2']
                }
        },
'chains': ['A']
}
```

Ici, 'A' est l'identifiant de l'unique chaîne de la molécule et '-3' est l'identifiant de l'unique résidu. Naturellement, nous manipulerons des molécules contenant plusieurs chaînes et plusieurs résidus par la suite, mais je vous conseille de commencer à manipuler notre petite arginine pour ne pas vous perdre dans les données !

4. Pour la séance 4, concentrez-vous sur le cœur de la fonction jusqu'à ce qu'elle réussisse à lire un "vrai" fichier PDB. Vous en avez un de disponible sur le serveur GitHub, et vous pouvez vous amusez à en trouver d'autres sur RCSB Protein Data Bank. Néanmoins, n'oubliez pas de structurer votre script proprement comme nous l'avons vu dans les séances précédentes ! Prenez cette habitude dès à présent : ceux qui pensent qu'il le feront après le font rarement ! De plus, vous aurez à utiliser cette fonction dans votre projet, donc autant faire les choses correctement dès maintenant.

Correction :

```
import string # Pour utiliser strip()

def parsePDBMultiChains(infile) :

    # lecture du fichier PDB
    f = open(infile, "r")
    lines = f.readlines()
    f.close()
        ### Comme on a chargé tout le fichier dans lines, on peut tout de suite le
            fermer

    # Initialisation
    dddd_PDB = {}
    dddd_PDB["chains"] = []

    # parcoure le PDB
    cmptAltLoc = False
    for line in lines :
        if line[0:4] == "ATOM" :

                        ### Pour gérer les différentes conformations possibles dans
                            un même fichier
            ### (ici, on choisi simplement la première lue)
            if cmptAltLoc == False:
                altLoc = line[17] # 'A', 'B', etc. ou juste ' '
                cmptAltLoc = True

            if line[17] == altLoc:

                chain = line[21]

                ### Si on ne teste pas, on écrasera toutes les données contenues
                    dans dPDB[chain] à chaque boucle
                if not chain in dddd_PDB["chains"] :
                    dddd_PDB["chains"].append(chain)
                    dddd_PDB[chain] = {}
                    dddd_PDB[chain]["reslist"] = []

                ### Ici, il est conseillé de nettoyer la chaîne de caractères :
                curres = line[22:26].strip()
                ### strip() retire les espaces avant et après une chaine de caractè
                    res

                ### Si on ne teste pas, on écrasera toutes les données contenues
                    dans dPDB[chain][curres]
                if not curres in dddd_PDB[chain]["reslist"] :
                    dddd_PDB[chain]["reslist"].append(curres)
```

```
                    dddd_PDB[chain][curres] = {}

                    ### Ici, il faudrait plutôt testé aussi le "Alternate location
                        indicator" pour éviter des problèmes...
                    dddd_PDB[chain][curres]["resname"] = line[17:20].strip()

                    dddd_PDB[chain][curres]["atomlist"] = []


                atomtype = line[12:16].strip()
                dddd_PDB[chain][curres]["atomlist"].append(atomtype)
                dddd_PDB[chain][curres][atomtype] = {}

                ### Mieux de convertir en float les valeurs de position :
                dddd_PDB[chain][curres][atomtype]["x"] = float(line[30:38])
                dddd_PDB[chain][curres][atomtype]["y"] = float(line[38:46])
                dddd_PDB[chain][curres][atomtype]["z"] = float(line[46:54])

                dddd_PDB[chain][curres][atomtype]["id"] = line[6:11].strip()

    return dddd_PDB
```

5. Pour aller un peu plus loin : La fonction précédente est fonctionnelle, mais devrait encore être améliorée (gestion des erreurs, description détaillée, tests, etc.). Vu que vous aurez à l'utiliser dans tous les projets, c'est l'occasion d'en faire une vraie fonction stable et bien décrite.

Correction : *Il y a plusieurs solutions et degrés de sophistication, mais a minima, cela devrait ressembler à ça :*

```
#!/ usr/bin/env python
#coding: utf-8

"""
Author: Arnaud Ferré
Contact: arnaud.ferre_at_u-psud.fr
Date: 20/02/2017
Description: Script containing many useful functions for parsing and processing PDF
    files (i.e. 3D structure of proteins)
Licence: DSSL (http://dssl.flyounet.net/)
"""

import string
import sys # Pour accéder à exit()



def parsePDBMultiChains(infile) :
    """
        Cette fonction permet de charger un fichier PDB (Protein Data Bank) au
            format ATOM.
        Puis de parser son contenu (structure 3D d'une molécule) pour le stocker
            dans une variable Python.

        Paramètre(s) :
            - infile : emplacement du fichier à charger et parser

        Valeur renvoyée :
            - dddd_PDB : dictionnaire complexe contenant les informations
                structurelle de la protéine étudiée. Celle-ci est décomposée en chaî
                nes, chaque chaîne est décomposée en résidus, et chaque résidu est d
                écomposé en atomes. Enfin, chaque atome possède des coordonnées cart
                ésiennes dans l'espace. On peut accéder également à la liste des
                composants en faisant : dddd_PDB["chains"], dddd_PDB[IDchain]["
                reslist"], dddd_PDB[IDchain][IDres]["atomlist"].

        Pour plus d'informations sur les fichiers PDB et en particulier le format
            ATOM, veuillez vous reporter à :
        http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html#
            ATOM
    """
```

```
    ### On vérifie que l'ouverture du fichier se passe correctement :
    try:
        f = open(infile, "r")
        lines = f.readlines()
        f.close()
    except:
        print("Le fichier n'a pu être chargé correctement. Vérifiez que le fichier
            existe bien et relancez votre programme.")
        sys.exit(0) ### Stoppe simplement l'exécution du programme.


    dddd_PDB = {}
    dddd_PDB["chains"] = []

    cmptAltLoc = False
    for line in lines :
        if line[0:4] == "ATOM" :

            if cmptAltLoc == False:
                altLoc = line[17]
                cmptAltLoc = True

            if line[17] == altLoc:

                chain = line[21]

                if not chain in dddd_PDB["chains"] :
                    dddd_PDB["chains"].append(chain)
                    dddd_PDB[chain] = {}
                    dddd_PDB[chain]["reslist"] = []

                curres = line[22:26].strip()

                if not curres in dddd_PDB[chain]["reslist"] :
                    dddd_PDB[chain]["reslist"].append(curres)
                    dddd_PDB[chain][curres] = {}
                    dddd_PDB[chain][curres]["resname"] = line[17:20].strip()
                    dddd_PDB[chain][curres]["atomlist"] = []

                atomtype = line[12:16].strip()
                dddd_PDB[chain][curres]["atomlist"].append(atomtype)
                dddd_PDB[chain][curres][atomtype] = {}

                # On pourrait aussi vérifier que les chaînes de caractères
                    continnent bien des float pour éviter une erreur.
                dddd_PDB[chain][curres][atomtype]["x"] = float(line[30:38])
                dddd_PDB[chain][curres][atomtype]["y"] = float(line[38:46])
                dddd_PDB[chain][curres][atomtype]["z"] = float(line[46:54])

                dddd_PDB[chain][curres][atomtype]["id"] = line[6:11].strip()

    return dddd_PDB


###
# Ici, vous écrirez vos prochaines fonctions
###


### Ici, vous pourrez testez vos fonctions :
if __name__ == "__main__":

    # Pour afficher une structure de façon un peu plus esthétique :
    import json
    print("Données pur l'arginine : \n"+json.dumps(parsePDBMultiChains("arginine.pdb
        "), indent = 4))

    print("\nIdentifiants des chaînes de la protéine 1EJH :")
    print(parsePDBMultiChains("1EJH.pdb")["chains"])
    print("\nIdentifiants des résidus de la chaîne A de la protéine 1EJH :")
    print(parsePDBMultiChains("1EJH.pdb")["A"]["reslist"])
```

6. Et on utilise Git/GitHub!!! Mettez votre script sur notre GitHub à la place du fichier que vous

avez pushé dernièrement (dans testCommit). Essayons pour cela de respecter une convention de nommage : structureTools_PrenonNom.py. Pour la suite des TPs, utilisez et travaillez directement sur ce même fichier pour stocker vos fonctions intermédiaires (calcul du centre de masse, calcul du rayon de giration, RMSD, etc.). exécution

4 Un peu d'aide en pagaille !

1. "Parser", kesako ? C'est un anglicisme couramment utilisé en programmation informatique qui signifie : analyser la syntaxe. En pratique, cela signifie découper et extraire des informations d'une chaîne de caractères en les mettant dans un format exploitable par un programme.
2. Récupérer une sous-partie d'une chaîne de caractères : Exécutez le code suivant :

```
line = "azertyuiopqsdfghjklmwxcvbn"
print line
print line[4:10]
```

   Pensez à utilisez cette méthode plutôt que la méthode `split()`, car il n'y a pas toujours d'espaces entre 2 champs d'un fichier PDB !
3. Dictionnaire de dictionnaire de dictionnaire, etc. : Rappelons qu'un dictionnaire est une structure en Python qui, à une valeur non-mutable appelée clé, associe une valeur quelconque. Cette valeur peut donc être elle-même un dictionnaire ! Et ainsi de suite. Le code suivant est donc tout à fait valide :

```
dico = dict() #Initialisation d'un dictionnaire vide
dico["un"] = dict() #Initialisation d'une première valeur de ce dictionnaire, de clé
     la chaîne de caractères "un". Cette valeur est un dictionnaire vide.
dico["un"]["deux"] = dict()
dico["un"]["deux"]["trois"] = dict()
for i in ['a','b','c','d']:
    dico["un"]["deux"]["trois"][i] = 0

print dico
```

   Conseil : Lorsque vous manipulez ce genre de structure, pour ne pas vous y perdre, vous pouvez adopter une règle de nommage de variable qui indique combien de dictionnaires imbriqués contient une variable. Pour l'exemple précédent, on aurait pu par exemple écrire :

```
dddd_dico = dict()
```

4. "Alternate location indicator" : La colonne 17 d'un fichier PDB contient une lettre qui permet de proposer plusieurs possibilités de conformation d'une même molécule. Choisissez celle que vous voulez, mais pensez à n'utiliser que celle-ci lorsque vous parserez le reste du fichier.
5. La méthode strip() : Une des problématiques du parsing, c'est que l'on récupère fréquemment des espaces (i.e. des " ") non-constants. Cela a son importance, car, par exemple, si vous avez "key" et " key ", cela ne sera pas considéré comme étant la même chaîne. Vous pourriez donc vous retrouvez à créer 2 clés distinctes dans un dictionnaire sans le vouloir. La méthode strip() permet justement d'éliminer les espaces parasites aux extrémités d'une chaîne de caractères. Vous pouvez tester le code suivant pour constater son effet :

```
line = "  abc defg   "
print ">>>"+line.strip()+"<<<"
```

6. Attention à ne pas écraser vos dictionnaires ! Lorsque vous faites ceci :

```
dddd_dico = dict()
```

   Si la variable `dddd_dico` contenait quelque chose, tout sera effacé ! Faites donc bien attention à initialiser vos dictionnaires en dehors des boucles (sauf si cela à un intérêt bien sûr), car c'est souvent comme ça que l'on fait cette erreur.
7. La première chose à tester sur une ligne est de savoir si c'est une ligne "ATOM" ou non !
8. N'oubliez pas que, en Python, vous pouvez tester simplement si une valeur appartient ou non à une liste de la manière suivante :

```
l_liste = [1,2,3,5,6,7,8]
if 5 in l_liste:
    print("5 est dans la liste")
if 4 not in l_liste:
    print("4 n'est pas dans la liste")
```

De-même, n'oubliez pas également que l'on peut accéder à la liste des clés d'un dictionnaire en utilisant la méthode `keys()` :

```
d_dico.keys()
```