USER GUIDE

OVERVIEW

User documentation enables a user to understand the features and on how to use a script. The three approaches for writing an instruction manual includes a guide with a step by step approach for a beginner, a thematic approach for an intermediate user and a list or a reference approach for an advanced user.

In this guide, the users are directed by a tutorial approach to an understanding of how the `pdb.py` script file can be used. Further, the functions included in the Pdb class are briefly explained with drawbacks.

Using the script `Pdb.py` you will be able to perform the atom-atom distance calculations, average B-factor calculations, extracting the Fasta sequence and ligand section of a pdb file.

This script takes a pdb file as an input from the command line and assigns it to a file object which is further used by the functions `get_Distance()`, `get_Bfactor()`, `get_Sequence` and `get_Ligand()`.

HARDWARE or SOFTWARE REQUIREMENTS

- Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM.
- Disk space: 2 to 3 GB.
- Operating systems: Windows® 10, macOS*, and Linux*

PREREQUISITES

The script requires the following to run:

- Python 3, including modules**:** math, statistics, argparse

INSTALLATION

To download the latest version of python3 for the operating system running on your computer use the link https://www.python.org/downloads/.

After the installation of python3 for your operating system you can check the version of the python installed by using the which command on the terminal.

```
$ which python3

/usr/bin/python3
```

INPUT SPECIFICATION

The command line accepts one input pdb file along with the script file. Later, this input file is assigned to a file object in the pdb class and used by the functions `get_Distance()`, `get_Bfactor()`, `get_Sequence()`, and `get_Ligand()`.

TUTORIAL

The first step is to install the latest version of python3. You can find the directions for installing python3 for the operating system running on your computer by following the link from the installation section.

In this tutorial, we will run the code from the command line of the UNIX shell. Thus we expect users to be familiar with running the code from the terminal. There are many graphical user interfaces (GUI) such as Jupyter notebook and IDEs available for running a python code, however, running code in the terminal is necessary for performing tasks such as to input a file from the command line.

To follow along, first create a folder in the home directory named `python_scripts`. Next download `pdb.py` script, 1CXC.pdb file which is provided as a test file and place them in the `python_scripts` directory.

To run the scripts, navigate to your newly created directory `~/python_scripts`. Run the command line python3 with `pdb.py` and `1CXC.pdb` files to calculate the atom-atom distances, average B-factor for all atoms, main chain atoms and CA atom. Furthermore, to extract the sequence and the ligand section of a particular chain from a pdb file. You will find the output for the distance calculations to be written to a text file with the input atom names separated by an underscore. Moreover, you will find the sequence file as a Fasta format and ligand information as a pdb file with the input chain name, in your `python_scripts` directory. However, the output for calculating the average B-factor is written to the terminal.

```
#to make a directory

$ mkdir python_scripts

#to navigate to directory python_scripts

$ cd ~/python_scripts

##run pdb.py for calculating atom-atom distance, average B-factor
#
##for all atoms, main chain and Ca atom. Additionally, if needed
#
##to find the sequence or to extract the ligand information of a
#
##chain in a pdb file.

## Function 1: calculating the atom-atom distances for input atoms

###run pdb.py to calculate the atom-atom distances
```

example usage: run with pdb file, and the output file is written to the present directory

~/python_scripts $ python3 pdb.py 1CXC.pdb --distance

This will generate an output on the terminal

#You will need to input the first atom and second atom in the terminal
#
#to perform the atom-atom distance calculations.
#
#The expected output after running the script pdb.py if the input for
#
#the first atom is CA and the second atom is C for distance calculations.

Input the first atom: CA
Input the second atom: C
The atom-atom distance calculations are written to CA_C.txt file

This will generate an output file:

This output file can be found within the python_scripts directory

<output>CA_C.txt
Calculated atom-atom distance with residue name, chain ID, residue number and distance.

##Function 2: Calculating the average B-factor for all atoms, main chain atoms and CA atom

###run pdb.py to calculate the average B-factor for all atoms, main chain atoms and CA atom

example usage: run with pdb file

~/python_scripts $ python3 pdb.py 1CXC.pdb --bfactor

This will generate an output on the terminal:

Average overall B-factor is: 19.834
Backbone averaged B-factor is: 18.442
Average alpha carbon B-factor is: 18.38

##Function 3: obtaining the fasta sequence for input chain ID

###run pdb.py to extract the fasta sequence

example usage: run with pdb file, and the output file is written to the present directory

```
~/python_scripts $ python3 pdb.py 1CXC.pdb --sequence

This will generate an output on the terminal:

#You will need to input the chain id  in the terminal to extract the
#
#sequence.
#
#The expected output after running the script pdb.py if the input for
#
#the chain is A.

This will generate an output file:

This output file can be found within the python_scripts directory

<output>A.fasta
# Amino acid sequence in fasta format for the input chain

## Function 4: extracting the ligand section for input chain ID

###run pdb.py to extract the ligand section of the pdb filename

example usage: run with pdb file, and the output file is written to
the present directory

~/python_scripts $ python3 pdb.py 1CXC.pdb --ligand

This will generate an output on the terminal:

#You will need to input the chain id  in the terminal to extract the
#
#ligand information.
#
#The expected output after running the script pdb.py if the input for
#
#the chain is A.

This will generate an output file:

This output file can be found within the python_scripts directory

<output>A.pdb
# Ligand section of the pdb file

## The fasta or pdb file will be empty if the input chain is not
present in the pdb file.
```

```
Full list of options:

…

usage: pdb.py [-h][--distance][--bfactor][--sequence][--ligand]

positional arguments:
  filename    input a pdb file

optional arguments:
  -h, --help  show this help message and exit
  --distance  distance calculates the atom-atom distances
  --bfactor   bfactor calculates the average B-factor for all
              atoms, main chain atoms and CA atom
  --sequence  sequence extracts the fasta sequence
  --ligand    ligand extract the ligand section of the pdb file
```

DESCRIPTION

A class named Pdb is created with the functions `get_Distance()`, `get_Bfactor()`, `get_Seqeunce()` and `get_Ligand()` that take in a pdb file from the user which is specified when the script is run from the terminal and assigns it to a variable. Moreover, when assigning to a variable it reads the lines by using `readline()` method to return a list containing lines. Further, this variable is used to iterate over the lines in the pdb file and `split()` method is applied to return a list of words split by space. Thus, this would allow accessing each column in a pdb file.

`get_Distance()` returns atom-atom distance of a pdb file, by the receiving the first and second atom as an input from the user. A conditional statement is used to access the rows starting with ATOM. The X, Y, and Z coordinates of the input atoms are appended to two empty lists that are initialized. Furthermore, the atom number, residue name, residue number and chain id are also appended to the lists. Moreover, these lists are iterated to calculate the Euclidean distance between input atoms of all the residues in the pdb file by importing the math module and combining the data using `zip()` function. The output is written to a file `atom1_atom2.txt` (example: if the input atoms where CA and C the output file would be `CA_C.txt`) by formatting the output using the string method `center()` that returns the string aligned center with a fixed length. Further, the distance between any two different atoms could be calculated by giving different atom names as an input.

`get_Bfactor()` returns overall Bfactor for all-atom, main chain atoms and CA atom of the pdb file. A conditional statement is used to access the rows starting with ATOM and B-factor values of all atoms in the pdb file are appended to an initialized empty list. Further, the B-factor values for main chain atoms (CA, C, N, O) and all CA atom are also appended to two separate empty lists that were initialized. These lists are used to calculate the overall B-factor, the average B-factor for the main chain and for CA atom by importing the statistics module and applying the `mean()` method to each list. Further, the output is returned to the terminal.

`get_Sequence()` function returns a Fasta file with the sequence for the chain ID specified by the user as an input. A conditional statement is used to access the rows that start with SEQRES and the rows that have the input chain ID. Further, the SEQRES section of the pdb file is appended to a list. Later, the list is converted to a string and to a single letter code of an amino

acid using a dictionary which has three letter code as keys and single letter code of amino acid as values. Additionally, after converting the residues to a single letter, the sequence is written to a file in Fasta format. Moreover, the sequences of the other chains in a pdb file can be extracted by specifying different chain ID as an input.

`get_Ligand()` function returns the ligand section of input pdb file, by receiving chain ID as an input from the user. A conditional statement is used to access the rows starting with HETATM and to exclude the water molecules. Furthermore, the extracted HETATM section is written to a file by formatting the output using the string methods `rjust(), centre (), and ljust()` to have a fixed string length for each column in the file. `rjust()` method returns the string right justified in a specified string length, `ljust()` returns the string left justified in a specified string length and `centre ()` returns by aligning a string centre, using a fixed length. The extracted ligand information can be visualized using visualization tool such as Pymol. Further, the ligands of the other chains in a pdb file can be extracted by specifying different chain ID as an input.

EXAMPLES

1. get_Distance()

a) A small part of the code from the class Pdb which calculates the atom-atom distances.

```
def get_Distance(self):
  """Returns all the distance between the C and CA atoms of a pdb file"""
  #readlines() method to the pdb file to return a list containing lines
  all_lines = self.pdb_file.readlines()
  # empty list to hold the X, Y, Z coordinates, atom number, residue name, chain and residue number of the Ca atom
  atom_1 = list()
  # empty list to hold the X, Y, Z coordinates, atom number, residue name, chain and residue number of the C atom
  atom_2 = list()
  # to get first atom input from the user
  atom1 = input("Input the first atom: ")
  # to get second atom input from the user
  atom2 = input("Input the second atom: ")
  # for loop to iterate over the lines in the pdb file
  for i in all_lines:
    #split() method to return a list of words splitted by space
    j = i.split()
    #condition to specify the row starting with ATOM and the row where the alpha carbon atom is present
    if j[0]=='ATOM'and j[2]== atom1:
        #appends the atom_number, residue name, chain id, residue number,X,Y,Z coordinates of Ca to an empty list
        atom_1.append((j[1],j[3],j[4],j[5],float(j[6]), float(j[7]),float(j[8])))
    #condition to specify the row starting with ATOM and the row where the carbon atom is present
    if j[0]=='ATOM'and j[2]== atom2:
        #appends the atom_number, residue name, chain id, residue number,X,Y,Z coordinates of C atom to an empty list
        atom_2.append((j[1],j[3],j[4],j[5],float(j[6]), float(j[7]),float(j[8])))
  #calculation of the euclidean distance between the Ca and C atom
  #
  #zip function is mainly used to combining data of two iterable elements ca_atom and c_atom
  dist_calc = [( a[0],a[1],a[2],a[3],(a[4]-b[4])**2, (a[5] - b[5])**2, (a[6] - b[6])**2) for a, b in zip(atom_1, atom_2) ]
  ca_c_dist = [(atm_no,res,chain,res_num,round(math.sqrt( x+y+z),3)) for atm_no, res,chain,res_num,x,y,z in dist_calc ]
  #output file to write the calculated distances between Ca and C atom
  with open(atom1+'_'+atom2+".txt","w") as dist_file:
   # writing the headings to the distance.txt file
   dist_file.write("Distance between all the "+atom1+ " and "+atom2+" are: "+"\n"+"Residue"+"\t "+"Chain"+"\t"+"Residue_num"+"\t"+"Distance"+"\n")
    #writing the calculated distance to a file name distance.txt
   for atm_no, res, chain,res_num,distance  in ca_c_dist:
    #center() method will center align the string, with the specified length
    dist_file.write(res.center(0)+chain.center(8)+ res_num.center(8)+(str(distance)).center(12)+"\n" )
  print("The distance between Ca and C atom are written to "+atom1+"_"+atom2+".txt file")
  return
```

b) The expected output of the `get_Distance()` function written to a terminal to input atoms CA and C.

```
Input the first atom: CA
Input the second atom: C
The atom-atom distance calculations are written to CA_C.txt file
```

c) The expected output of the `get_Distance()` function written to a file CA_C.txt for the input atoms CA and C.

```
Distance between all the CA and C are:
Residue   Chain   Residue_num Distance
    GLN         A           1    1.493
    GLU         A           2     1.52
    GLY         A           3    1.515
    ASP         A           4    1.514
    PRO         A           5     1.53
    GLU         A           6    1.515
    ALA         A           7     1.52
    GLY         A           8    1.521
    ALA         A           9    1.519
    LYS         A          10    1.506
    ALA         A          11    1.527
    PHE         A          12    1.533
    ASN         A          13    1.499
    GLN         A          14    1.519
    CYS         A          15     1.52
    GLN         A          16     1.52
    THR         A          17    1.516
```

2. **`get_Bfactor()`**

a) A small part of the code from the class Pdb that calculates the average B-factor for all atoms, backbone and Ca atoms in a pdb file.

```
def get_Bfactor(self):
  """Returns the average overall B-factor, the backbone average B-factors and average alpha-carbon B-factors"""
  # readlines() method to the pdb file to return a list containing lines
  all_lines = self.pdb_file.readlines()
  # empty list to hold the bfactor values of all atoms
  avg_bfactor = list()
  # empty list to hold the bfactor values of main chain atoms
  mainchain_b_factor = list()
  # empty list to hold the bfactor values of Ca atom
  ca_bfactor = list()
  # for loop to iterate over the lines in the pdb file
  for i in all_lines:
   j=i.split()#split() method to return a list of words splitted by space
   if j[0]=='ATOM':#condition to specify the row where there is atom
       b_fac = (float(j[10])) # extracts the bfactor values of all atoms
       avg_bfactor.append(b_fac) # appends the bfactor values of all atoms to an empty list
   if j[0]=='ATOM'and (j[2]=='CA')|(j[2]=='C')|(j[2]=='N')|(j[2]=='O'):#condition to specify the row which starts with ATOM and rows where there are backbone atoms
       b_factor_mainchain = (float(j[10]))# extracts the bfactor values of main chain atoms
       mainchain_b_factor.append(b_factor_mainchain)# appends the bfactor values of main chain atoms to an empty list
   if j[0]=='ATOM' and (j[2]=='CA'):#condition to specify the row starting with ATOM and the row where the alpha carbon atom is present
       b_fac_ca = (float(j[10])) # extracts the bfactor values of Ca atom
       ca_bfactor.append(b_fac_ca) # appends the bfactor values of Ca atom to an empty list
  print (" Average overall B-factor is: "+ str(round(statistics.mean(avg_bfactor),3)))
  print (" Backbone averaged B-factor is: "+str(round(statistics.mean(mainchain_b_factor),3)))
  print (" Average alpha carbon B-factor is: "+str(round(statistics.mean(ca_bfactor),3)))
  return
```

b) The expected output of the `get_Bfactor()` function written to the terminal after the execution of the `pdb.py` script.

```
Average overall B-factor is: 19.834
Backbone averaged B-factor is: 18.442
Average alpha carbon B-factor is: 18.38
```

3. **`get_Sequence()`**

a) A small part of the code from the class Pdb of the `pdb.py script` that returns a Fasta file with the sequence for the chain ID specified by the user as an input.

```
def get_Sequence(self):
 # readlines() method to the pdb file to return a list containing lines
 all_lines = self.pdb_file.readlines()
 # dictionary with key as three code of amino acid and values as a single letter code
 amino_dict = {'CYS': 'C', 'ASP': 'D', 'SER': 'S', 'GLN': 'Q', 'LYS': 'K',
    'ILE': 'I', 'PRO': 'P', 'THR': 'T', 'PHE': 'F', 'ASN': 'N',
    'GLY': 'G', 'HIS': 'H', 'LEU': 'L', 'ARG': 'R', 'TRP': 'W',
    'ALA': 'A', 'VAL':'V', 'GLU': 'E', 'TYR': 'Y', 'MET': 'M',}
 # empty list to hold SEQRES section of the pdb file
 seqres = list()
 # creating an input statement from the user to obtain the chain id
 chain = input('Input the chain id for which for the seqeunce :')
 # for loop to iterate over the lines in the pdb file
 for i in all_lines:
       #condition to specify the row which starts with SEQRES
       if i.startswith('SEQRES'):
           #split() method to return a list of words splitted by space
           lines = i.split()
           #condition to specify rows for the input chain ID
           if lines[2]== chain :
             # appends the SEQRES for the input chain to an empty list
             seqres.append(lines[4:])
 #converting seqres list of list to a single list
 flat_list = [residue for sublist in seqres for residue in sublist]
 #converting list to a string
 amino_3_letter = ''.join(flat_list)
 #converting the amino acid three letter to a single letter
 amino_3_to_1 = "".join(amino_dict[amino_3_letter[x:x+3]] for x in range(0, len(amino_3_letter), 3))
 #extracting the PDB three letter code by using the input file
 PDB_ID = self.pdb_file.name[:4]
 #writing the seqeunce for the input chain to chain.fasta file
 with open(chain+".fasta","w") as seq_file:
  seq_file.write(">"+PDB_ID+":"+chain+"|"+"PDBID|CHAIN|SEQUENCE"+"\n"+amino_3_to_1)
 return
```

b) The expected output of the `get_Sequence()` function written to a terminal to input chain ID A.

```
Input the chain id for which for the seqeunce :A
The fasta sequence is written to A.fasta file
```

c) The expected output of the `get_Sequence()` function written to `A.fasta` file which can be found within the `python_scripts` directory given the input chain ID as A.

```
>1CXC:A|PDBID|CHAIN|SEQUENCE
QEGDPEAGAKAFNQCQTCHVIVDDSGTTIAGRNAKTGPNLYGVVGRTAGTQADFKGYGEGMKEAGAKGLAWDEEHFVQYVQDPTKFLKEYTGDAKAKGKMTFKLKKEADAHNIWAYLQQVAVRP
```

4. `get_Ligand()`

a) A small part of the code from the class Pdb of the `pdb.py` script that returns a pdb file with ligand information for the chain ID specified by the user as an input.

```
def get_Ligand(self):

 """Returns a file with the ligand information of the pdb file with the chain"""

 # readlines() method to the pdb file to return a list containing lines
 all_lines = self.pdb_file.readlines()

 # creating an input statement from the user to obtain the chain id
 chain = chain = input('Input the chain id for the ligand information :')

 # output file to write the ligand information from the pdb file
 with open(chain+".pdb","w") as lig_file:

  # for loop to iterate over the lines in the pdb file
  for i in all_lines:

   #split() method to return a list of words splitted by space
   j=i.split()

   # condition to specify the row starting with HETATM and the row where there no HOH (water) present
   if j[0]=='HETATM' and j[3] != 'HOH' and j[4]==chain:

    # center() method will center align the string, with the specified length, rjust() method to return a string right justified, ljust() method to return a string
left justified
    lig_file.write(( j[0].ljust(8)+j[1].ljust(5)+j[2].center(4)+j[3].ljust(5))+j[4].rjust(1)+j[5].rjust(4)+ str('%8.3f' % (float(j[6]))).rjust(8)+ str('%8.3f' %
(float(j[7]))).rjust(8)+str('%8.3f' % (float(j[8]))).rjust(8)+str('%6.2f'%(float(j[9]))).rjust(6)+str('%6.2f'%(float(j[10]))).ljust(6)+j[11].rjust(12)+'\n')
  print("The ligand from the pdb file for input chain is written to "+chain+".pdb file")
  return
```

b) The expected output of the `get_Ligand()` function written to a terminal.

```
Input the chain id for the ligand information :A
The ligand from the pdb file for input chain is written to A.pdb file
```

c) **The expected output of the `get_Ligand()` function** written to `A.txt` file which can be found within the `python_scripts` directory given the input chain ID as A.

```
1 HETATM  921  CHA HEM  A 125  82.996  25.963  14.866  1.00 10.23           C
2 HETATM  922  CHB HEM  A 125  81.309  21.716  16.390  1.00  9.29           C
3 HETATM  923  CHC HEM  A 125  83.694  22.281  20.550  1.00 11.47           C
4 HETATM  924  CHD HEM  A 125  84.964  26.739  19.207  1.00 10.59           C
5 HETATM  925  C1A HEM  A 125  82.377  24.725  14.891  1.00  9.03           C
6 HETATM  926  C2A HEM  A 125  81.637  24.150  13.781  1.00  9.51           C
7 HETATM  927  C3A HEM  A 125  81.184  22.943  14.193  1.00  7.83           C
8 HETATM  928  C4A HEM  A 125  81.617  22.796  15.572  1.00  8.84           C
9 HETATM  929  CMA HEM  A 125  80.466  21.920  13.324  1.00  8.08           C
```

DRAWBACKS

Even though we know pdb files are absolutely formatted, some files have their own disadvantages. Thus, the function `get_Bfactor()` will not be able to calculate the average B-factor of a pdb file which is incorrectly formatted meaning the pdb files that are not formatted with proper spacing.

```
HETATM 4345  C1  NAG A   1      13.880  62.184  55.731  1.00 99.05           C
HETATM 4346  C2  NAG A   1      14.221  63.447  54.944  1.00 98.56           C
HETATM 4347  C3  NAG A   1      15.476  64.132  55.475  1.00 99.46           C
HETATM 4348  C4  NAG A   1      15.435  64.234  56.993  1.00100.00           C
HETATM 4349  C5  NAG A   1      15.115  62.872  57.588  1.00100.13           C
HETATM 4350  C6  NAG A   1      15.118  62.950  59.107  1.00100.12           C
HETATM 4351  C7  NAG A   1      13.415  63.230  52.666  1.00 95.41           C
HETATM 4352  C8  NAG A   1      13.592  62.529  51.356  1.00 95.30           C
HETATM 4353  N2  NAG A   1      14.407  63.114  53.545  1.00 96.90           N
HETATM 4354  O3  NAG A   1      15.583  65.424  54.923  1.00 99.71           O
HETATM 4355  O4  NAG A   1      16.673  64.696  57.487  1.00100.07           O
HETATM 4356  O5  NAG A   1      13.852  62.463  57.115  1.00 99.74           O
```

```
ValueError: could not convert string to float: '1.00100.00'
```

`get_Bfactor()` function cannot work with `3fgr.pdb` file, because the function initially splits the column by space using the `split()` method. The 3fgr.pdb file has a spacing error (Figure 1), thus the average B-factor could not be calculated.

CONTACT

If you have any problems, questions, ideas or suggestions, please contact us by mailing to nhk4@student.le.ac.uk.