Biol 366 – Introduction to Bioinformatics

Mini-project – 15%

Sameer Patel – 20428838

April 20th 2015

Introduction

For my mini-project I decided to learn the programming language python and set out to make a tool that determines the X, Y, Z coordinates and the temperature factor (B value) of any given PDB file.

Python is a clear and powerful object-oriented programming language that uses an elegant syntax, comes with a large standard library that supports common programing tasks such as connecting to webservers and is easily extended by adding new pre-made modules to help you.

In this assignment, I will be using a software called Enthought Canopy. Enthought Canopy is a comprehensive Python analysis environment that provides easy installation of the core scientific analytic and scientific python packages, creating a robust platform to develop on. I added in *modules* such as: *Numpy* 1.8.1-3, *Scipy* 0.15.1-1 and the most important *biopython* 1.65-1.

Numpy is an extension to pyton, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

Scipy is a python based ecosystem of open-source software for mathematics, science and engineering in particular. They are: NumPy, SciPy library, Matplotlib, IPython, Sympy, and pandas.

Biopython is a set of freely available tools, it is a distributed collaborative effort to develop python libraries and applications which address the needs of current and future work in bioinformatics. Unfortunately this tool is only available for computers that have 32 bits, thus we use Enthought Canopy to set up a virtual environment to write our code in.

PDB stand for the Protein Data Bank, the Protein Data bank is an archive of experimentally determined 3D structures of large biological molecules, including proteins and nucleic acids. The PDB archive is available at no cost to users and is updated weekly.

Each PDB file contains: PDB identifiers, date deposited, methods used, NMR simulation, compound molecule/keywords, PDB authors, primary literature references, other literature references, Sequence of DNA chain D, Sequence of protein chains, Heteroatom declarations, locations of where the author thinks the helices & beta sheets are, crystallographic information, Nucleic acid atom residue coordinates, Amino acid atom residue coordinates, heteroatom atom coordinates, water atom coordinates, and some odds and end information to wrap it all up.

Let's break down what the script does and what does it all mean. In simple terms: the script written uses pre-made classes and modules to achieve our goal of determining the X, Y, Z coordinates and the Temperature factor (B-value) from a PDB file.

```
import Bio
```
→We are importing the Bio module from Enthought Canopy's packages and all the Classes that are associated with it. Bio Module is BioPython itself.

```
from Bio import PDB
```
→ Now we give specific instructions to fetch the PDB module from Biopython package and import PDB module and gives access to all of its definitions and values.

```
pdb1 = PDB.PDBList()
```
→ PDB.PDBList() reads as we are calling the PDBList file in the PDB folder and assigning it to pdb1. PDBList() is a class that provides quick access to the structure lists on the PDB server or its mirrors. It also provides a function to retrieve PDB files from the server. PDBList() class has its own list of methods that we can use to perform other functions.

```
pdb1.retrieve_pdb_file("1C7D")
```
→ We are using previously assigned pdb1 and calling the methods within the PDB.PDBList() module. We are calling the retrieve_pdb_file method. This method retrieves a PDB structure file from the PDB server and stores it in a local file tree. Then creates a PDB-style directory tree. The argument given is the name of the compressed PDB file you wish to fetch from the PDB server.

```
parser = PDB.PDBParser(PERMISSIVE=1)
```
→ PDB.PDBParser() reads as we are calling the PDBParser file in the PDB folder and assigning it to parser. PDBParser() is a class that parses the retrieved PDB files and returns a structure object. PDBParser() class has its own list of methods that we can use to perform other functions. Permissive=1 is there to evaluate as a Boolean function and 1 means it is true.

```
structure =
parser.get_structure("1C7D",r'C:\Users\Sameer\AppData\Local\Enth
ought\Canopy\User\Lib\site-packages\Bio\c7\pdb1c7d.ent')
```

→ We are using previously assigned parser and calling the methods within the PDB.PDBParser() module. We are calling the get_structure() method. This method retrieves a PDB structure. It takes in two arguments. **Id** – string, the id that will be used for the structure. **File** – name of the PDB file or the location to where the downloaded PDB file is located if it isn't in your default directory. The r' is there to indicate that you are reading said file.

```
print(structure)
```

→ After retrieving the structure, we use a print statement to get our script to display our structure's values that the program can then manipulate.

```
for model in structure:
    for chain in model:
        for residue in chain:
            for atom in residue:
```

→ This is known as a nested for loop. We are iterating over multiple lists (structures, model, chain and residue) very quickly.

```
                N = atom.get_name()
                I = atom.get_id()
                Y = atom.get_coord()
                V = atom.get_vector()
                O = atom.get_occupancy()
                B = atom.get_bfactor()
```

→ We are calling the atom object's methods and assigning each function to a variable. The Atom object stores atom name (both with and without spaces), coordinates, B-factor, occupancy, and alternative location specifier. The Coordinates type: float, B-factor type: number.

```
            if 10 < B < 50:
                        print(V, B)
```

→ This is a type of conditional statement. It is an if statement. We if followed by a condition and end the first line with a colon. There follows a block of indented lines that will only be executed if the condition above is true. In our case we want to print(V,B) ONLY IF B value is bigger than 10 and less than 50.

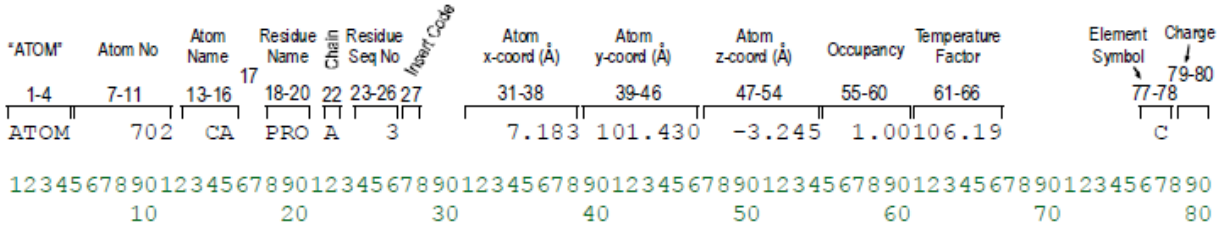
Code written in Python 3.4.1

This is what the code looks like with the sample input being the1C7D.pdb file in .ent format. The file is called pdb1c7d.ent and is located in \Bio\c7 folder.

```
# -*- coding: utf-8 -*-
import Bio
#first we import the Bio Module from BioPython
from Bio import PDB
pdb1 = PDB.PDBList()
pdb1.retrieve_pdb_file("1C7D")
parser = PDB.PDBParser(PERMISSIVE=1)
structure =
parser.get_structure("1C7D",r'C:\Users\Sameer\AppData\Local\Enth
ought\Canopy\User\Lib\site-packages\Bio\c7\pdb1c7d.ent')
print(structure)
for model in structure:
    for chain in model:
        for residue in chain:
            for atom in residue:
                N = atom.get_name()
                I = atom.get_id()
                Y = atom.get_coord()
                V = atom.get_vector()
                O = atom.get_occupancy()
                B = atom.get_bfactor()
                if 10 < B < 50:
                    print(Y, B)
#print(dir(structure))
#print(dir(model))
#print(dir(chain))
#print(dir(model))
#print(dir(residue))
#print(dir(atom))
```

The # means that they are just the comments and will not be interpreted by python. The last 6 lines are there for reference if one forgets the methods available to them for each of the classes.

What the raw PDB file looks like and the official coordinate file format



ATOM 1 N MET A 1 20.241 31.402 42.660 1.00 78.88 N
ATOM 2 CA MET A 1 20.518 30.010 42.996 1.00 67.71 C
ATOM 3 C MET A 1 22.010 29.710 42.917 1.00 54.70 C
ATOM 4 O MET A 1 22.542 28.976 43.752 1.00 77.38 O
ATOM 5 CB MET A 1 19.746 29.059 42.076 1.00 73.69 C
ATOM 6 CG MET A 1 19.080 27.902 42.804 1.00 74.20 C
ATOM 7 SD MET A 1 19.417 26.302 42.031 1.00157.91 S
ATOM 8 CE MET A 1 17.800 25.045 43.461 1.00 45.71 C
ATOM 9 N LEU A 2 22.693 30.273 41.919 1.00 38.79 N
ATOM 10 CA LEU A 2 24.136 30.053 41.840 1.00 32.10 C
ATOM 11 C LEU A 2 24.897 31.166 42.545 1.00 28.23 C
ATOM 12 O LEU A 2 24.936 32.300 42.063 1.00 30.93 O
ATOM 13 CB LEU A 2 24.585 29.948 40.381 1.00 29.35 C
ATOM 14 CG LEU A 2 24.123 28.688 39.644 1.00 35.45 C
ATOM 15 CD1 LEU A 2 24.450 28.774 38.161 1.00 39.25 C

And this is what the output looks like when we run the code. Keep in mind, this is a very small bit of the output. The X, Y and Z coordinates are put into a vector format!.

(<Vector -2.17, 5.04, 47.70>, 41.48)
(<Vector -2.41, 24.34, 26.27>, 46.89)
(<Vector -16.24, 4.24, 45.33>, 33.42)
(<Vector -9.55, -0.08, 23.60>, 26.15)
(<Vector 11.35, 3.78, 21.67>, 28.44)
(<Vector -22.86, 7.23, 34.93>, 27.38)
(<Vector -12.73, 15.91, 24.50>, 38.88)
(<Vector -1.21, 10.66, 40.02>, 40.28)
(<Vector -13.60, 16.89, 33.31>, 40.61)
(<Vector -15.28, 6.93, 50.87>, 37.6)
(<Vector -18.62, 3.04, 29.59>, 40.2)
(<Vector -6.14, -3.63, 28.30>, 38.73)
(<Vector 14.58, 0.24, 22.11>, 36.83)
(<Vector 1.55, 7.34, 39.24>, 40.01)
(<Vector 6.00, 3.89, 35.89>, 49.53)

Definitions:

Argument: A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function.
Body: The sequence of statements inside a function definition.
Conditional statement: A statement that controls the flow of execution depending on some condition.
Condition: The Boolean expression in a conditional statement that determines which branch is executed.
Compound statement: A statement that consists of a header and a body. The header ends with a Colon (:). The body is indented relative to the header.
Class: A user-defined type. A class definition creates a new class object.
Class object: An object that contains information about a user-defined type. The class object can be used to create instances of the type.
Floating-point: A type that represents numbers with fractional parts.
Function: A named sequence of statements that performs some useful operation. Functions may or may not take arguments and may or may not produce a result.
Function object: A value created by a function definition. The name of the function is a variable that refers to a function object
Function call: A statement that executes a function. It consists of the function name followed by an argument list.
Loop: A part of a program that can execute repeatedly.
Module: A file that contains a collection of related functions and other definitions.
Import statement: A statement that reads a module file and creates a module object.
Module object: A value created by an import statement that provides access to the values defined in a module.
Method: A function that is associated with an object and called using dot notation.
Object: Something a variable can refer to. For now, you can use “object” and “value” interchangeably.

Object-oriented programming: A style of programming in which data and the operations that manipulate it are organized into classes and methods.
Parse: To examine a program and analyze the syntactic structure.
Print statement: An instruction that causes the Python interpreter to display a value on the screen.
Relational operator: One of the operators that compares its operands: ==, !=, >, <, >=, and <=.
Script: A program stored in a file (usually one that will be interpreted).
Syntax: The structure of a program.
String: A type that represents sequences of characters.
Statement: A section of code that represents a command or action. So far, the statements we have seen are assignments and print statements.
Type: A category of values. The types we have seen so far are integers (type int), floating-point numbers (type float), and strings (type str).
Value: One of the basic units of data, like a number or string, that a program manipulates.
Variable: A name that refers to a value.

.**References:**

Biopython, I., & Biopython, S. (n.d.). Bioinformatics using Python for Biologists 9.1, 1–14.

Callaway, J., Cummings, M., & Deroski, B. (1996). Protein Data Bank contents guide: Atomic coordinate entry format description. *Brookhaven National Laboratory*. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Protein+Data+Bank+Contents+Guide+:+Atomic+Coordinate+Entry+Format+Description#2

Chang, J., Chapman, B., Friedberg, I., Hamelryck, T., Hoon, M. De, Cock, P., … Wilczy, B. (2010a). Biopython Tutorial and Cookbook. *Update*, *2010*(November), 15–19. http://doi.org/10.1145/360262.360268

Chang, J., Chapman, B., Friedberg, I., Hamelryck, T., Hoon, M. De, Cock, P., … Wilczy, B. (2010b). Biopython Tutorial and Cookbook. *Update*, *2010*(November), 15–19. http://doi.org/10.1145/360262.360268

Chang, J., Chapman, B., Friedberg, I., Hamelryck, T., Hoon, M. De, Cock, P., … Wilczy, B. (2010c). Biopython Tutorial and Cookbook. *Update*, *2010*(November), 15–19. http://doi.org/10.1145/360262.360268

Chapman, B. (2009). Biopython Installation. *Administrator*, 1–10.

Hamelryck, T., & Manderick, B. (2003). PDB file parser and structure class implemented in Python. *Bioinformatics*, *19*(17), 2308–2310. http://doi.org/10.1093/bioinformatics/btg299

Jones, M. (2013). Python for Biologists, 231. Retrieved from http://pythonforbiologists.com/index.php/version/

Like, T., & Scientist, C. (n.d.). Think Python. *Xtemp-01*.

Repressor, L. a C., Complex, D. N. a, Bell, C. E., & Lewis, M. (n.d.). Anatomy of a PDB file. *Gene*, 1–4.

Universitetsparken, C., Project, T. B., Biopython, T., & Struct, P. (2005). The Biopython Structural Bioinformatics FAQ. *Molecular Biology*, *2*(0), 96–108. http://doi.org/10.1016/j.meegid.2011.03.022