CS483 Assignment #1

Molecular Visualization and Python

Due date: Thursday Jan. 22 at the start of class.
Hand in on Tuesday Jan. 20 for 5 bonus marks.

General Notes for this and Future Assignments:

Chimera will be used as a "workbench" for the various assignments in the course. The primary purpose of the Chimera application is to display macromolecules in a way that illustrates various features of molecular structure and its relation to biological activity. There are three ways for users to interact with the structure being displayed:

1. using menu invocations,
2. typing commands into the command line text box at the bottom of the Chimera window (use menu item: **Favorites/Command Line**),
3. executing a Python script (start with menu item: **Tools/General Controls/IDLE**).

For now, you should assume that *all* Chimera related questions will require menu invocations and/or Python scripts. You will only use the command line facilities to change the display when execution of a Python script cannot accomplish the needed change in the display. For example, it is recommended that your script will set the background colour to white. This will save printer ink when you print out images of the molecular display. Since your script will typically need to start with a blank display you will also want to close the previous session. This can be done using **File/Close Session** in the Chimera window. However, both the closing of the previous session (if there was one) and the setting of the background can be done using the following code which should be at the start of any script:

```
import chimera
from chimera import runCommand
runCommand("close session; set bgColor white")
```

Handing in assignments: Answers to various questions, well documented Python scripts, and graphics (which need not be in colour) should be submitted in paper form so that the TA can provide feedback. In addition to this, each assignment will require that you **email the following files to the TA** so that they can be run or viewed by the TA:

- any Python script that is part of the answer (send the script via email so that it can be executed by the TA, and submit a duplicate of the script as hardcopy so that the TA can comment on the script), [**Be sure to provide appropriate documentation with the script**]
- session files specified by a question,
- occasionally, a .png colour graphics file will be required if a session file cannot provide the needed colour display.

Students should take care to properly identify all submitted files so that the TA can properly relate any file to the appropriate question.

Purpose of this assignment:

- Work with macromolecular visualization tools (as a preparatory step you should get familiar with the Chimera application and the documentation that is available).
- Use a Python script that will generate molecular models and then modify their visual display.

Marks:
1:[5 exercises: 5* 3 = **15**], **2**:[**5**], **3**:[parts (a) & (b) 7 + 8 = **15**], **4**:[script: 12 + output: 4 = **16**], **5**:[script for (a): 12 + output for (a): 4 + script for (b): 12 + output for (b): 4 = **32**].
The assignment will be marked out of 80.

"Warm up" Questions

1) Do Exercise 1 from Chapter 1 of the text (Section 1.5): Answers should be submitted as Session files (.py). Send them as email attachments to the TA. It is expected that all displays will be in colour.
2) Do Exercise 3 from Chapter 1 of the text (Section 1.5). Include a list of the amino acid names shown in the figure.

Python Scripts

3) **Two simple Python scripts:**

(a) **Listing residue hydrophobicity values**: Get a Python Shell and use it to get a new window in which you will write a Python script. The script should read a PDB file when given the PDB ID of a protein (the ID can be a parameter string in the program). The script should fetch the PDB file and then print (to the shell window) a table with a single row for each standard residue in the protein. A standard residue is one of the 20 residues that are typically found in a protein. Each line should contain the residue type, followed by the residue position and chain identifier (the latter two entries separated by a period). A final column should print out the Kyte-Doolittle hydrophobicity for that residue (note that this is an attribute for a residue object). Here is the first output line for the protein with PDB ID = 1QU9:

```
SER   2.A  -0.8
```

Notes:

- You should use Python output formats to get a consistent format for the lines of output.
- If "residues" have type "HOH" or if they correspond to ligands then they should be ignored in this printout.

- Some observations related to 1QU9: If you look at the sequence for this protein, you will see that it starts with MET, but there are no atom coordinates for this MET residue. Consequently, it is not in the residues list for the protein and your output will start with SER.

(b) **Colour residues using hydrophobicity values**: Chimera has various formats to specify colours in a display. See the discussion about *color_name* at the webpage: http://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/color.html. One of the formats for specifying a colour is by means of an RGB tuple where RGB is an acronym for Red Green Blue. Here are some sample colours and their RGB specifications:

white	(1.000, 1.000, 1.000)
dodger blue	(0.118, 0.565, 1.000)
orange red	(1.000, 0.271, 0.000)

For this exercise you should write a Python function with definition header:
`def residueKDcolour(r)`
where `r` is a residue object. The function should return a tuple that specifies a colour corresponding to the Kyte-Doolittle hydrophobicity for that residue. If you consult the webpage at:
https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/hydrophob.html
you will notice that kdHydrophobicity ranges from 4.5 (most hydrophobic) down to -4.5 (least hydrophobic, in fact, most hydrophilic). Here are the rules for generation of the color tuple:

- kdHydrophobicity equal to 0.0 should correspond to the color white with a tuple value of (1.000, 1.000, 1.000).
- kdHydrophobicity equal to 4.5 should correspond to the color orange red with a tuple value of (1.000, 0.271, 0.000).
- kdHydrophobicity equal to -4.5 should correspond to the color dodge blue with a tuple value of (0.118, 0.565, 1.000).
- kdHydrophobicity in the interval (0.0, 4.5) should produce a colour that is intermediate between white and orange red, in other words, it should be a linear interpolation between (1.000, 1.000, 1.000) and (1.000, 0.271, 0.000).
- kdHydrophobicity in the interval (-4.5, 0.0) should produce a colour that is intermediate between dodger blue and white, in other words, it should be a linear interpolation between (0.118, 0.565, 1.000) and (1.000, 1.000, 1.000).

Note: In Python tuples cannot be used like vectors but it is easy to convert tuples to arrays and arrays back to tuples. The array class can be imported from numpy.

To test your function, you will write a mainline script that uses the raw_input function to get a four character PDB ID from the user. Your script will use this string in the `chimera.openModels.open` function call to fetch the protein file. Then go through all the residues in the protein and change the colour of each residue in the display. The colour is to be derived from the `residueKDcolour` function. The colour of a residue can be changed by using a runCommand invocation. Consider the execution of

```
runCommand("color 0.00,1.00,0.00 #0:5")
```

This statement will change the colour of the residue at position 5 in model 0 to green. Notice that the colour tuple does not use brackets and there are no spaces among the 3 numbers. Note also that 5 is the position and not the index of the residue. In the loop that progresses through the residues, your script should generate the character string that is used as input for `runCommand`. During execution of the script you should see all the colour changes being made.

To check whether your script has generated the correct colours you can use raw_input to ask the user for permission to execute the final statement which will be:

```
runCommand("surface")
```

Chimera will generate a surface around the protein and the colours of the surface will be inherited from the colours that you have given the residues. In another Chimera window you can use the menus to fetch the same protein and then invoke the menu item: **Presets/Interactive 3 (hydrophobicity surface)**. Your surface should look the same as this surface. So, your program will duplicate existing Chimera functionality but it will give you some practise in Python programming and it can provide starting code if you want to colour a surface using some other residue property that is different from hydrophobicity.

Test your final script on the protein with PDB ID 1MBN. This is sperm whale myoglobin. If your surface colours are correct you will see that the porphyrin ring is embedded in binding site that is very hydrophobic.

4) **Cysteine bridges**: It is very common in Structural Bioinformatics to simply collect statistics related to some protein property or functionality. Write a script that starts by reading a text file containing a list of PDB identifiers each with a chain ID suffix. The first line in the input file will contain some descriptive text that identifies the content for the user and should be printed out by the script just before the output described below (it is otherwise ignored). Here is a short sample file:

```
Venoms
  1F8UB
  1A3FA
  1ND1A
  4E0VA
  1POBA
  1POCA
  1WQ8A
  3GBOA
  3C05A
  3C05B
  1FCQA
  1BQYA
  1ATLA
  1SEGA
  1KUGA
  1FASA
  1QNXA
  4AEIA
  1XT3A
  2F9RA
  1NXBA
  1DTXA
  1UOSA
  1UOSB
  2H8UA
  1ANSA
  3VC0A
```

Use the following statements to get the path name of the input file:

```
import tkFileDialog
filePathName = tkFileDialog.askopenfilename()
```

While reading this file the script should fetch each protein and then process the residues in the indicated chain. When all the chains have been accessed and processed the script should print out the following information using an output format of your own choosing (making sure that all the output is properly described).

- Total number of protein chains processed.
- Total number of standard residues processed in the specified chains.
- Percentage of the chains containing at least one CYS residue.
- Percentage of the chains containing at least one CYS residue that is involved in the formation of a disulphide bridge.
- Percentage of the chains containing at least one CYS residue that is involved in the formation of a disulphide bridge AND at least one other CYS residue that is **not** involved in a disulphide bridge.
- Percentage of residues (considering all the chains) that are CYS residues.
- Percentage of residues (considering all the chains) that are CYS residues involved in the formation of a disulphide bridge.

When printing a percentage use only 2 decimal places of accuracy (%6.2f).

Run the script using the first 1000 entries from PDBselect. (http://bioinfo.mni.th-mh.de/pdbselect/) which I have extracted and set up in the data file `PDBselect1000`. During debugging of your code you might want to deal with a much smaller input file. Caution: it will take quite a lot of time to fetch these files…

Run the script again on the list of venoms presented above and note the increase in the percentage of disulphide bridging. This is typical of animal venoms which are usually small proteins that use cysteine bridges to enhance both their stability and the rigidity of a particular conformational shape. The last entry 3VC0 is a venom protein that comes from the Inland Taipan found in Australia (http://en.wikipedia.org/wiki/Inland_taipan). Its neurotoxin venom is estimated to be 50 times more powerful than cobra venom and typically causes a very quick death.

5) **Side chain H-bonds**:

a) You can use Chimera to display hydrogen bonds. For example, after fetching the PDB file 1RKI you can use the menu invocation: **Tools/Structure Analysis/FindHBond** to get a dialog window that allows you generate the display of all the hydrogen bonds. To simplify the display, you should perform **Select/Residue/all nonstandard** followed by **Actions/AtomsBonds/delete** to eliminate hydrogen bonds involving water molecules and various ligands. You will see several hydrogen bonds that are responsible for the formation of secondary structure (both helices and beta sheets). If you invoke the menu item **Actions/AtomsBonds/show** you will notice that there are several other hydrogen bonds that involve atoms in the side chains (not in the back bone).

Python statements to get hydrogen bonds is fairly easy. Consider the statements:

```
import FindHBond
hb_L = FindHBond.findHBonds([prot], distSlop = 0.4, angleSlop =
20.0)
```

The list `hb_L` is a list of 2-tuples. The first entry in a 2-tuple is an atom object representing the hydrogen bond donor and the second entry is an atom object representing the acceptor. After

executing these statements you can extract the tuples that meet the requirements described in the next paragraph. Note that execution of these statements will get the list of H bonds but the display is not affected.

The script for this exercise should ask the user for a PDB ID. It will then fetch that protein and print out a list of atom pairs such that the first atom of the pair is a hydrogen bond donor and the second atom is a hydrogen bond acceptor with the restriction that we will ignore a pair if either atom is in a nonstandard residue or if both atoms are in a backbone. Atoms should be specified by residue name, residue position, chain ID and atom name. Separate the atom specifications with four dashes. For example, here are the first few lines that I got in the processing of 1RKI:

```
ARG 19.A NH2  ----  GLU 15.A OE2
ARG 95.A NH2  ----  ILE 69.A O
ARG 96.A NH2  ----  GLU 72.A OE2
ARG 95.B NH1  ----  ASP 71.B OD1
ARG 96.B NH1  ----  GLU 72.B OE2
```

b) For this exercise, we are interested in collecting statistics that will help us to understand how often we can expect to see hydrogen bonds involving atoms in sidechains. Near the start of your script you should declare three dictionaries:

```
resTypeCount_D = {}
donorAtomCount_D = {}
acptrAtomCount_D = {}
```

After this, the script will read a file that contains a list of PDB identifiers each followed by a chain ID suffix (same type of input as in Exercise 4 above). We will refer to the chain selected by this suffix as the **designated** chain.

Use the following statements to get the path name of the input file:

```
import tkFileDialog
filePathName = tkFileDialog.askopenfilename()
```

While reading this file the script should fetch each protein in the list and for each protein it will compute a list of hydrogen bonds meeting the same constraints as described in Part (a). We will refer to this list as the **sidechain HBond** list. We now describe the significance of the dictionaries:

For `resTypeCount_D` a key will be a residue type (for example, "ASN") and the final value corresponding to the key will be the number of these residues that have been encountered while going through all the designated chains. The count does not include residues in the other chains. You should end up with 20 entries in this dictionary.

For `donorAtomCount_D` a key will be a string representing the donor atom name specified as the residue type followed by the atom name separated by a space, for example: "`ARG NH2`". After processing all the designated chains, the final value corresponding to a key will be the number of times such a donor atom appears on the donor side of a pair in the sidechain HBond list. Your script should ignore any donor atom that is not in the designated chain.

For `acptrAtomCount_D` a key will be a string representing the acceptor atom name specified as the residue type followed by the atom name separated by a space, for example: "`GLU OE2`".

After processing all the designated chains, the final value corresponding to a key will be the number of times such an acceptor atom appears on the acceptor side of a pair in the sidechain HBond list. Your script should ignore any acceptor atom that is not in the designated chain.

Some examples (assuming the designated chain is "A"):
The sidechain HBond list entry: `ARG 95.A NH2 ---- ILE 69.A O` would cause an increment of the value for the key `ARG NH2` in the `donorAtomCount_D` dictionary and an increment of the value for the key `ILE O` in the `acptrAtomCount_D` dictionary. Now suppose the hydrogen bond is between chains:
The sidechain HBond list entry: `ARG 85.A NH2 ---- ILE 59.B O` would cause an increment of the value for the key `ARG NH2` in the `donorAtomCount_D` dictionary but the value for the key `ILE O` in the `acptrAtomCount_D` dictionary is not incremented because this acceptor is not in the designated chain. Note that an entry in the HBond list such as: `ARG 96.B NH1 ---- GLU 72.B OE2` would be ignored since both donor and acceptor are not in the designated chain.

After all the chains have been processed print out three tables with the following titles:

Amino Acid Percentages
This table will have 20 rows. Each row has two entries: the residue type (for example, "`ASN`") followed by the frequency of that residue in all the designated chains expressed as a percentage. Note: you will have to calculate the total number of residues that are in all the designated chains. The rows should be ordered alphabetically with respect to the residue type.

Donor Atom Percentages
This table will have a row for each donor atom encountered. Each row has two entries: the donor atom name (for example: "`ARG NH2`") followed by the frequency of that donor expressed as a percentage of the total number of times its residue is in the designated chains. For example, the percentage for donor "`ARG NH2`" is calculated as:
`donorAtomCount_D["ARG NH2"]*100.0/resTypeCount_D["ARG"].`

Acceptor Atom Percentages
This table will have a row for each acceptor atom encountered. Each row has two entries: the acceptor atom name (for example: "`GLU OE2`") followed by the frequency of that acceptor expressed as a percentage of the total number of times its residue is in the designated chains. For example, the percentage for acceptor "`GLU OE2`" is calculated as:
`acptrAtomCount_D["GLU OE2"]*100.0/resTypeCount_D["GLU"].`

All percentages should have 4 decimal precision (%7.4f).

Test your code using the input file PDBselectDimerList.txt.

Start early. **This is NOT an assignment that can be done the night before the due date!**