Getting Started with Chimera and Python

Finding Chimera

- You download Chimera from the UCSF website:
- http://www.cgl.ucsf.edu/chimera/
- If you need help with the menus, here is a useful link:
  - http://www.cgl.ucsf.edu/chimera/docs/UsersGuide/frametut.html
  - Then click on "Getting Started - Menu Version"
    - For now, you can avoid a lot of the entire tutorial (only the menu version is currently relevant to the first assignment).

Chimera & Python

- Python is part of the Chimera download.
- To get a Python Shell you can use the menu item: Tools/General Controls/IDLE.
  - To make this invocation a bit faster, you can set up an "IDLE button" on your Toolbar by checking the appropriate box after clicking the "**Add Tool Icon…**" on first page of the Chimera application.
  - The Python Shell has multiple uses:
    - You can test syntax or execution of short Python scripts.
    - You can use the Shell window to get output results from a script that is in execution.
      - You can also provide input to a running script.
    - You can open an editing window that may be used to generate a new Python script or to modify a recently produced script.

Python Scripts

- Editing of new scripts can be initiated from the Shell with: File/New Window.
- Any script typed into this editor can be saved using the File menu of the editor window.
- Later, you can get that same script by using File/Recent Files in the Python Shell window.
- To run your script, you should save it and then use the Run/Run Module of the edit window.
- If you have print statements in the script (especially important for debugging), the print output will be directed to the Python Shell window.

Python Scripts for Chimera

- Start your script with: `import chimera`
- To load a PDB file that is resident on your disk use a full path name.
  - For example to get file 1k4c.pdb stored in directory `Temp`:

```
my_mod=chimera.openModels.open('C:\\Temp\\1crn.pdb',type="PDB")
```

OR

- To load a PDB file from the RCSB use the PDB id:

```
      my_mod = chimera.openModels.open('1crn', type="PDB")
```

  - Note that "`.pdb`" is not used in this case.
- The variable `my_mod` will be a list of open models.
  - For a PDB file this list will usually have a single element because the file contains only one molecule.
    - Some NMR derived PDB files contain several models (Examples: 2K9Z, 2L1T, 2KTS)
  - Note: Some files, such as .sdf files, can contain several molecules.
    - Incidentally: a list of open models will be accessible in the Chimera window by using Favorites/Model Panel.

The Chimera Object Hierarchy (1)

- You can experiment with the Chimera object hierarchy by using the Python Shell to fetch the file for crambin from the PDB:

```
>>> import chimera
>>> openModels = chimera.openModels.open('1crn', type="PDB")
>>>
```

  - Hitting the Enter key after typing the second line, causes Chimera to fetch 1crn from the PDB and the protein is displayed in the Chimera window.
- By accessing the first member (index 0) of the open models list we derive an object that is a protein molecule:

```
>>> prot = openModels[0]
```

- Note that if we now type `prot` followed by a period:

```
>>> prot.
```

  we get a rather long popup list of all the attributes for this object.
    - Use the up/down arrow keys to go through the list of attributes.

The Chimera Object Hierarchy (2)

- When the molecule accessed from the open models list is a protein, then most of our interactions with the Chimera hierarchy will make use of the following relationships:
  - a protein molecule contains a list of residues
  - a residue contains a list of atoms.
- Continuing our example, you can access the `i`th residue object in `prot` by using:

```
>>> a_res = prot.findResidue(i)
```

- To get a named atom in that residue, for example, the alpha carbon:

```
>>> ca_atom = a_res.findAtom('CA')
```

- To go to the next residue use:

```
next_res = prot.residueAfter(a_res)
```

Protein Chains

- Unfortunately, there is no chain object!
  - If necessary you could build your own chain object…
- It is possible to determine the chain in which a residue resides:

```
>>> prot.findResidue(44).id.chainId
```

Atoms in Chimera

- Atoms can be accessed directly (without going through the residues):

```
>>> my_atom = prot.atoms[i]
```

- For any atom object, you can get the coordinates of that atom:

```
>>> my_atomCoords = my_atom.coord()
```

- This will be a Point object. To get:
  - the *x*-coordinate use: `my_atomCoords[0]`
  - the *y*-coordinate use: `my_atomCoords[1]`
  - the *z*-coordinate use: `my_atomCoords[2]`.

Dealing with Point Objects

- You can import the Point class definition for your own use:

```
>>> from chimera import Point
```

- Then you can define a point object:

```
>>> q = Point()
```

- Coordinates in the point object can be changed, for example:

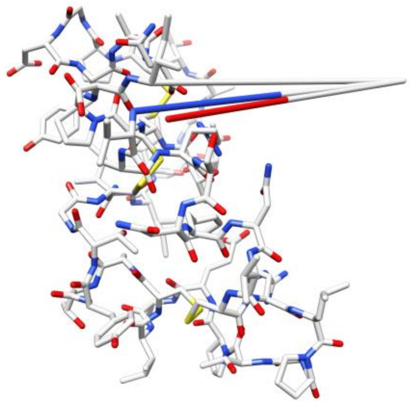
```
>>> q[0] = 22.000
```

- To change atom coordinates, define the contents of a Point, say `q`, and then use:

```
>>> my_atom.setCoord(q)
```

Changing Atom Coordinates

- Here is Crambin with the coordinates of atom[33] set to (22.00, 0.00, 0.00) by using the techniques described on the previous slide.
  - There is considerable "bond stretch" because of this arbitrary change but Chimera does its best to display the altered structure.
    - Note: A ribbon diagram will "smooth out" this single atom deviation.



Other Useful Methods and Attributes

- There are typically several methods associated with classes in Chimera.
- The next few slides look at a small (hopefully useful) subset of the methods available for various Chimera classes.
- There are several methods that relate to the display of a residue or atom but we will, for the most part, ignore these functions since the course mainly concentrates on geometric properties of the molecules being studied.

Methods and Attributes (protein objects)

- `prot.bonds`
  - Gives a list of bonds in the protein.
- `prot.sequence(chainID)`
  - Gives the amino acid sequence for the chain specified by the character held in chainID.
  - Note: you will have to know the chain ID characters before using this.
- `prot.sequences()` returns all non-trivial sequences.
- `prot.sequences(True)` returns all non-trivial sequences as a dictionary.

Methods and Attributes (residue objects) (1)

- Suppose `r` is a residue object. For example, using the previous code to define `prot`, we can extract the first residue with: `r = prot.residues[0].`
- `r.atomsNames()` returns a set of character strings that are the names of the atoms in the residue.
- `r.atoms` returns a list of objects each representing an atom in this residue.
- `r.chi1` float representing the Chi 1 angle.
  - There are other similar attributes: `chi2`, `chi3`, `chi4`.
- `r.findAtom(atomName)` returns an atom object when given a character string representing the name of an atom in this residue.

Methods and Attributes (residue objects) (2)

- `r.id.chainId` returns a character string representing the chain identifier of the chain containing this residue.
- `r.id.position` returns the position number of the residue in the chain.
  - Position numbers are specified by the PDB file.
  - This is usually different from the index of that residue in the list provided by `prot.residues`.
- `r.id.sameChain(another_res.id)` returns `True` if and only if both `r` and `another_res` correspond to residues in the same chain.
  - Note that the argument of the function is the ID of the residue.
- `r.isHelix` returns `True` iff `r` is in a helix.
- `r.isSheet, r.isStrand,` and `r.isHet` have corresponding functionality.
  - `isHet` returns `True` when the "residue" is a ligand or water molecule.

Methods and Attributes (residue objects) (3)

- `r.numAtoms()` returns the number of atoms in the residue.
- `r.phi, r.psi` returns the phi and psi dihedral angles for the residue.
- `r.type` returns the type of the residue (for example, 'ARG').

Methods and Attributes (atom objects) (1)

- Let us assume that an atom object `a` has been created, for example: `a = prot.atoms[0]`.
- `a.bonds` returns a tuple containing the bond objects for that atom.
- `a.connectsTo(another_atom)` returns a bond object if atom `a` is connected to the atom specified by `another_atom`.
- `a.coord()` returns a Point object representing the coordinates of atom `a`.
- `a.coordIndex()` returns the index of the atom within the atoms list. In this case it would return the integer 0 because we computed `a` using the statement: `a = prot.atoms[0]`.
- `a.findBond(another_atom)` returns the bond between `a` and `another_atom`.

Methods and Attributes (atom objects) (2)

- `a.idatmType()` returns the Chimera atom type for this atom.
  - Atom types are described in: http://www.cgl.ucsf.edu/chimera/docs/UsersGuide/idatm.html
- `a.idatmType()` returns a character string representing the name of the atom in the residue, for example, 'C3'.
- `a.neighbors` returns a list of atoms that are bonded to atom `a`.
- `a.residue` returns the residue that contains atom `a`.
- `a.setCoord(p)` sets the coordinates of `a` to the Point value `p`.
  - Note that we can convert a 3-tuple to a Point object: `a.setCoord(Point(1., 3., 4.))` sets the coordinates of `a` to (1., 3., 4.). Do not forget to first import the Point class. Place
    ```
    from chimera import Point
    ```
    near the start of the script.

Methods and Attributes (bond objects)

- Let us assume that a bond object `b` has been created, for example: `b = prot.bonds[0]`.
- `b.atoms` returns a tuple containing the two atoms at either end of this bond.
- `b.contains(an_atom)` returns `True` iff the `an_atom` object is at either end of bond `b`.
- `b.findAtom(atomIx)` returns the atom object indexed by `atomIx`.
  - Since two atoms specify a bond, `atomIx` should be 0 or 1.
- `b.length()` returns the floating point distance between the two atoms at either end of the bond.
  - `b.sqlength()` returns the square of this distance.
- `b.otherAtom(an_atom)` returns the atom that is at the other end on the bond containing the atom designated by `an_atom`.

NumPy: Numerical Python

- You can import a package to do scientific computing.
- Place the following line at the start of the program:

```
import numpy
```

- A NumPy tutorial is available at: http://numpy.scipy.org
  - Since the Chimera download includes NumPy, you can ignore the links that deal with getting NumPy and installing NumPy.
- NumPy will be very important when we do exercises that involve linear algebra.
  - Various examples will be given later in the course.