**PyFIREBALL Beta.**

***The atom class****.*

In PyFIREBALL Beta, as in solid state physics, one very important class of objects is the atom. The atom class derives much of it’s properties from the species class, and under-the-hood, there are many properties of the atom that are directly the species. The atom also contains it’s position in xyz, and a number of other methods and properties as outlined below. There are a number of unique features of the atom class that should be highlighted, for example, the subtract method in python has been overridden in the atom class so as distance is returned when you call:

atom1 – atom2

think of this as the line between those two atoms.

Also an atom can be moved by using the ‘+’ operator, this adds a list in the form of ratom [x, y, z] to its position.

Another feature is how very powerful the neighbor methods are for the atom class. The atom object can literally tell you what other elements are around it and (by recursion) around it’s neighbors also. This means that the atom object knows it’s immediate environment intimately to a very large distance and analysis can be carried out on this trivially (for example, if you want to know how much the atom’s been affected by an impurity that’s 2NN, 3NN or even 10NN away, you can probe that information).

The atom initializer requires that it’s element, atom number within the study and it’s position be declared:

ratom = [1.23, 1.23, 1.23] -> this is the xyz coordinates.

atom1 = atom(1, Cu, ratom) -> Cu is an element as read in from the Fdata.

Optionally, the atom’s charge can also be declared on initialization, the keyword is Q, this is soon to be depreciated as the atom’s charge is now summed from the charge on each individual shell when required.

The print method called on atom returns a string in the format:

“atomNumber: 2 -> @ 1.23, 1.23, 1.23”

*atom() immutable variables:*

.ratom -> Returns a list (ratom in Fireball) of the atoms xyz coordinates.

.charge -> Returns the charge on the atom by summing the charge (Qcurrent) on each shell

. neutralCharge -> Returns the neutral charge on the atom

.chargeChange -> Returns the result of .neutralCharge - .charge

.symbol -> Returns the element name of the atom

.mass -> Returns the mass of the atom

.nZ -> Returns the atom’s element number from the periodic table

.nssh -> Returns the number of shells on the atom

.nssh\_PP -> Returns the number of PP shells on the atom

.norb\_max -> Returns the atom’s number of orbitals

.norb\_PP\_max -> Returns the atom’s number of PP orbitals

.atomicE -> Returns the atom’s atomic energy (from the Fdata)

.rcutoff -> Returns the atom’s cutoff

.rcutoff\_PP -> Returns the atom’s PP cutoff

.rcutoff\_max -> Returns the atom’s maximum cutoff

.rcutoffA -> Returns the atom’s cutoff in Angstroms

.rcutoffA\_max -> Returns the atom’s maximum cutoff in Angstroms

.xmass -> Returns the atom’s xmass from FIREBALL

.Zval -> Returns the atom’s Zval from FIREBALL

*atom() mutable variables:*

.atomNumber -> the atoms atomnumber, although atom is usually in a list, this is useful.

.x -> X coordinate

.y -> Y coordinate

.z -> Z coordinate

.shells[] -> List of shell objects associated with this atom

.shellsPP[] -> List of PP shell objects associated with this atom

*atom() internal methods:*

.neighbor(NUMBER) -> returns the atom object of the neighbor of that given number to the calling atom.

. neighbourCount(element, depth, ignorelist = []) -> Returns the number of neighbours of species element. Default depth (as in 1 is first nearest neighbors, 2 2nd NN, etc) is 1 and ignorelist is for recursion. If you want only second nearest neighbours, for example, use a list comprehension.

*PyFIREBALL methods that apply to atom():*

readBas(filename) -> Reads in a .bas file, returns a list of atom objects

density(atom) -> Returns the density of states for that atom

atom2DOS(atom, HOMO = 999): -> Returns a DOS object with a dictionary of lists (instead of the previous list of lists) for the contributions

atomDistance(atom1, atom2) -> Returns the distance between these two atoms, same effect as atom1 – atom2

NeighborCount(Nlist, basList, iatom, neighspecies, depth, ignorelist = []) -> Takes in a Neighbor List of the form output from ReadNeigh and a basis list of the form output by ReadBas, along with the atom concernd and the species we want and depth, returns the number of neighbors of that depth. For recursion, the ignorlist means any already-counted atoms are ignored.

NearestSpecList(Nlist, iatom, basList, neighSpecies, numneigh) -> Returns an ordered list (distance low-high) of neighbors. This method again uses the neighbor map, sweeps through to find the neighbors that are of species neighSpecies, works out their distance from the iatom, then sorts that list and returns the requested numneigh. Unlike Neighborcount, it does not look to neighbors-of-neighbors

basList2Object(atomList, elements) -> Accepts the atomList argument, which is the list of atoms from ReadBas(), returns a list of atom objects.

chargesList2atoms(atomList, charges) -> Accepts the list of atoms from basList2Object and adds the charges to each atom object.

neighList2atomObjects(atomList, neighList, xlm) -> Accepts a list of atom objects and the neighbors list from readNeigh, adds a list of atoms neighbors to each atom. This method also adds the xlm vectors to the atom object. This is required for the "atom.neighbour()" method.