Python Short Course
Lecture 1: Python Overview

Richard P. Muller

Materials and Process Simulation Center

Spring, 2000

Course Outline

- Lecture 1: Introduction to Python
- Lecture 2: Numerical Python
  - Matrix multiplies, diagonalization, Ax=b solves, matrix inversion, etc.
- Lecture 3: Object oriented programming
  - classes, instances, overloading, polymorphism, etc.
- Lecture 4: Graphics
  - Tk widgets, 3D graphics with OpenGL
- Lecture 5: Python as a Glue Language
  - Extending python, linking shared object libraries, etc.

Why I Like Python

- Writing readable code is easy
  - Natural syntax to commands
  - Indentation-consciousness forces readability
- Reusing code is easy
  - PYTHONPATH/import are easy to use
- Object-oriented programming is easy
  - Finally understand what all the C++/Scheme programmers are talking about!
- Close ties to C
  - NumPy allows fast matrix algebra
  - Can dump time-intensive modules in C easily
- "Everything I like about Perl, and everything I like about Matlab."

Using Python Interactively

- Start python by typing "python"
  - /usr/bin/python on all platforms
- ^D (control-D) exits

```
% python
>>> ^D
%
```

- Comments start with ‘#’

```
>>> 2+2  #Comment on the same line as text
4
>>> 7/3 #Numbers are integers by default
2
>>> x = y = z = 0 #Multiple assigns at once
>>> z
0
```

Running Python Programs

- In general

```
% python myprogram.py
```

- Can also create executable scripts
  - Make file executable:

```
% chmod +x myprogram.py
```

  - The first line of the program tells the OS how to execute it:

```
#!/usr/bin/python
```

  - Then you can just type the script name to execute

```
% myprogram.py
```

  - or

```
% myprogram.py > myoutput.txt
```

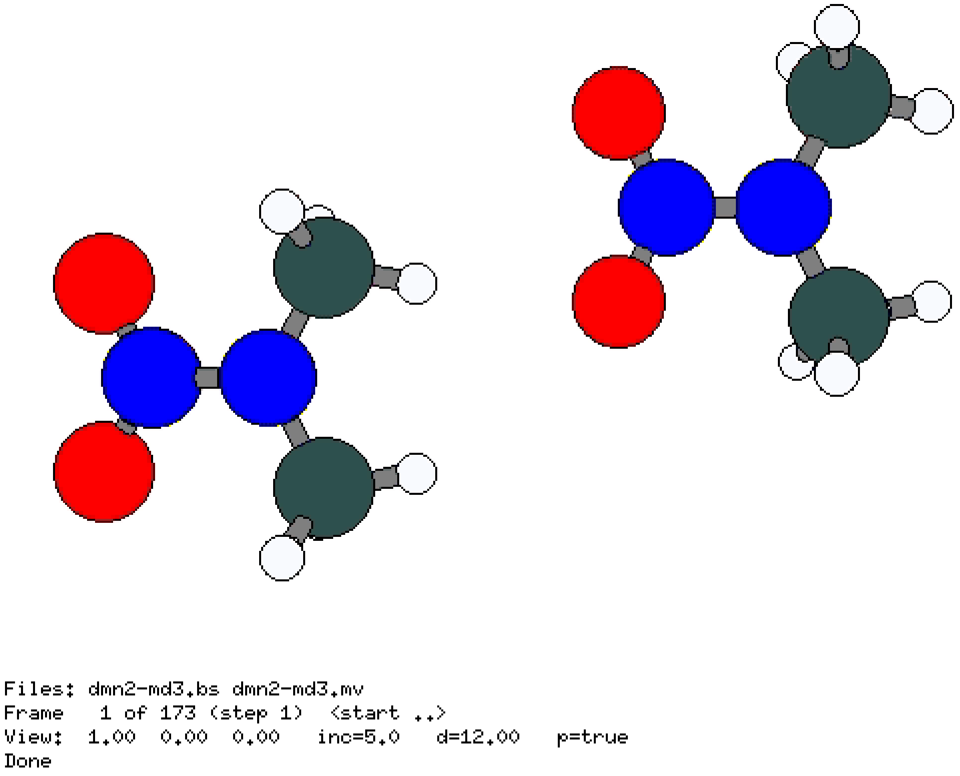
Setting up Emacs for Python

- There is a Python mode in Emacs which makes life much easier:
  - Indentation
  - Font coloring
- Instructions:
  - http://www.wag.caltech.edu/home/rpm/python_course/emacs_setup.html
  - or ask RPM for help
- There is also a Python development environment called IDLE which can be used on Windows/Mac.
  - We can install under X11 if people desire

Jaguar Geometry Optimization Viewer

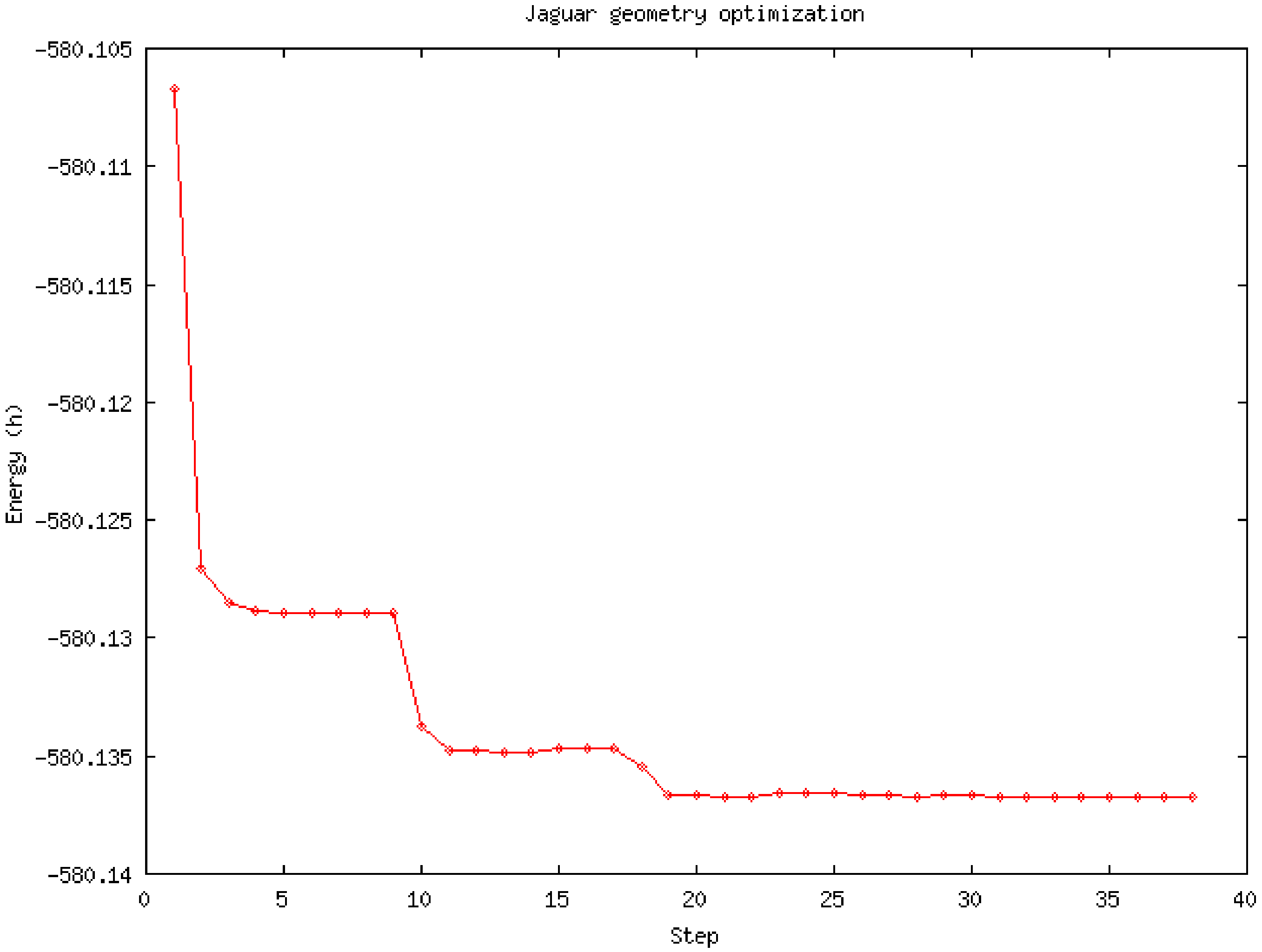
- Delve into an example of using Python
  - Plot energies in a gnuplot graphics window
  - Animate the geometry changes using XBS
- Do all of this in a program that is:
  - Easy to understand
  - Easy to modify
- Discuss Language issues as we come to them

Animation from XBS



```
Files: dmn2-md3.bs dmn2-md3.mv
Frame   1 of 173 (step 1)  <start ..>
View:  1.00   0.00   0.00    inc=5.0    d=12.00    p=true
Done
```

Output of Optimization Energies



Example: Jaguar output reader

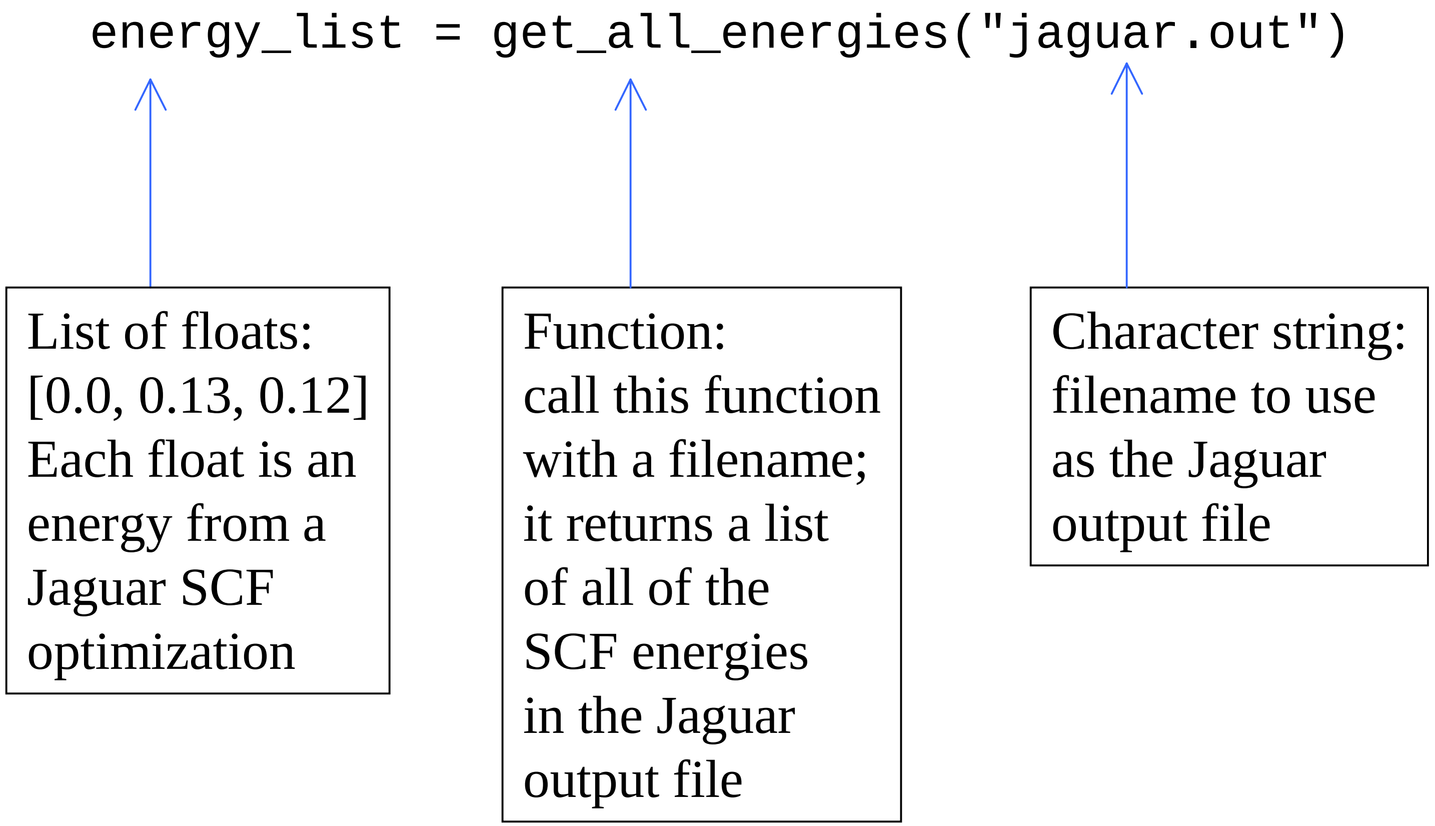
```
#!/usr/bin/python
import os
energy_list = get_all_energies("jaguar.out")
all_geometries = get_all_geos("jaguar.out")
write_xyz_file(all_geometries,"jaguar.xyz")
plot_energy_values(energy_list)
os.system("xbs jaguar.xyz")
```

Import statement

- import allows a Python script to access additional modules
- Modules
  - sys: stdin, stderr, argv
  - os: system, path
  - string: split
  - re: match compile
  - math: exp, sin, sqrt, pow

Strings, lists, floats

- First line of the program



```
energy_list = get_all_energies("jaguar.out")
```

List of floats:
[0.0, 0.13, 0.12]
Each float is an energy from a Jaguar SCF optimization

Function:
call this function with a filename; it returns a list of all of the SCF energies in the Jaguar output file

Character string:
filename to use as the Jaguar output file

Python Data Structures

- Strings

```
MyString = "this is a string"
myotherstring = 'this is also a string'
NewString = MyString + " " + MyOtherString
"If you mix quotes it doesn't end the string"
```

- Integers

```
A = 1    # Normal assignment
b = 3/5  #0, because of truncation
```

- Floats

```
pi = 3.1415927
```

Container Data Structures

- Containers hold collections of other data structures
- Lists
  - Most general sequence of objects
  - Can append, change arbitrary element, etc.

```
a = ['Hi',1,0.234]
```

- Tuples
  - On the fly data containers

```
atom = (atomic_symbol,x,y,z)
```

- Dictionaries
  - Text-indexed container

```
atomic_number = {'Dummy' : 0,'H' : 1,'He' : 2}
atomic_number['He']      # returns 2
```

Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam','eggs',100,1234]
>>> a[0]  # Lists start from 0, as in C
'spam'
>>> a[3]
1234
>>> a[-2] # Negative numbers index from the end
100
>>> a[:2] # ":" denotes a range
['spam','eggs']
```

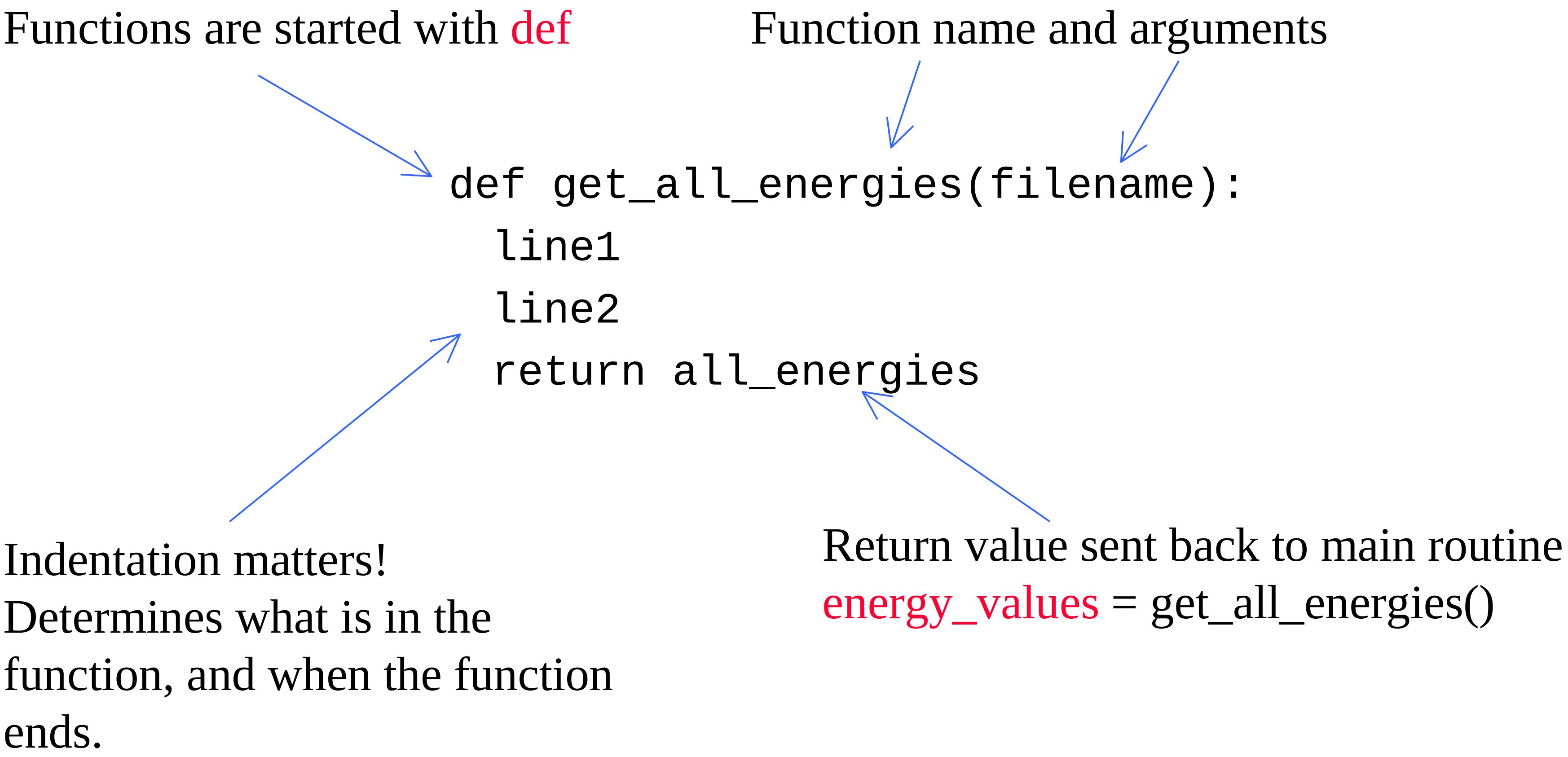
Adding to Lists

```
>>> a + ['bacon']
['spam','eggs',100,1234,'bacon']
>>> a.append('!')
['spam','eggs',100,1234,'!']
>>> 2*a
['spam','eggs',100,1234,'!','spam','eggs',100,
  1234,'!']
```

Example: Jaguar output reader

```
#!/usr/bin/python
import os
energy_list = get_all_energies("jaguar.out")
all_geometries = get_all_geos("jaguar.out")
write_xyz_file(all_geometries,"jaguar.xyz")
plot_energy_values(energy_list)
os.system("xbs jaguar.xyz")
```

Python functions



Functions are started with def

Function name and arguments

```
def get_all_energies(filename):
  line1
  line2
  return all_energies
```

Indentation matters! Determines what is in the function, and when the function ends.

Return value sent back to main routine
energy_values = get_all_energies()

get_all_energies function

define two new functions

Open/close file

```
def get_all_energies(filename):
  file = open(filename,'r')
  energies = []
  for line in file.readlines():
       if contains_etot(line):
              etot = extract_etot(line)
              energies.append(etot)
  file.close()
  return energies
```

file.readlines()
returns a list of
all of the lines in a file

Flow Control: Looping

- for and while statements can be used to control looping in a program:

```
colors = ['red','green','yellow','blue']
for color in colors:
  print color ' is my favorite color!'
```

- or

```
i = 0
while i < 10:
  print i         # Prints 0, 1, ..., 9
  i = i + 1       # No i++ in Python
```

for and range

- range returns a range of numbers

```
>>> range(3)
[0,1,2]
>>> range(1,3)
[1,2]
>>> range(2,5,2)
[2,4]
```

- for and range:

```
for i in range(10):
  print i                # Prints 0, 1, ..., 9
```

Regular Expressions

- Regular expressions are handled with the re module

```
import re
def contains_etot(line):
  return re.search("etot",line)
```

- Compiling a pattern makes searching faster

```
import re
etot_pattern = re.compile("etot")
def contains_etot(line):
  return etot_pattern.search(line)
```

- Regular expressions

`"^etot"`	Line beginning with "etot"
`"^[Ee]tot"`	Line beginning with "Etot" or "etot"
`"etot$"`	Line ending with "etot"

  - Many more examples: see re documentation at python.org

String manipulations

- String operations are handled with the string module

```
import string
def extract_etot(line):
  words = string.split(line)
  etot_str = words[6]
  etot = eval(etot_str)
  return etot
```

Split line into words based on whitespace

eval takes a string and turns it into a float

Recall Jag output line looks like:

```
etot   2   Y   N  6  M  -290.01543455332  2.4E-07  0.0+00  0.0E+00
```

Therefore, total energy is 6th element

Example: Jaguar output reader

- At last we've finished the first line of the code:

```
#!/usr/bin/python
import os
energy_list = get_all_energies("jaguar.out")
all_geometries = get_all_geos("jaguar.out")
write_xyz_file(all_geometries,"jaguar.xyz")
plot_energy_values(energy_list)
os.system("xbs jaguar.xyz")
```

- Look at how to extract all geometries from file:

```
all_geometries = get_all_geos("jaguar.out")
```

get_all_geos function

- Return all the geometries in a Jaguar geometry optimization output file

```
def get_all_geos(filename):
  file = open(filename,'r')
  all_geos = []
  while 1:
       line = file.readline()
       if not line: break
       if start_of_geo_pattern(line):
             geo = get_one_geo(file)
             all_geos.append(geo)
  return all_geos
```

Pattern searching similar to contains_etot()

New function

get_one_geo function

```
def get_one_geo(line):
  geo = []
  while 1:                        # Infinite loop
       line = file.readline()     # Only read one line
       if not line: break         # At EOF
       words = string.split(line) # Break into words
       if len(words) < 4: break   # Done with geo
       sym = words[0]
       x = eval(words[1])
       y = eval(words[2])
       z = eval(words[3])
       atom = (sym,x,y,z)         # Store atom in tuple
       geo.append(atom)
  return geo
```

Data structures for molecules

- Atoms are tuples

```
atom1 = ('H',0.0,0.0,0.0)
```

- Molecules are lists of atoms

```
h2o = [ atom1, atom2, atom3]
```

- all_geos are lists of molecules

```
trajectory = [h2o0,h2o1,h2o2,h2o3,h2o4]
```

Example: Jaguar output reader

- Two down:

```
#!/usr/bin/python
import os
energy_list = get_all_energies("jaguar.out")
all_geometries = get_all_geos("jaguar.out")
write_xyz_file(all_geometries,"jaguar.xyz")
plot_energy_values(energy_list)
os.system("xbs jaguar.xyz")
```

- Look at how to write geometries:

```
write_xyz_file(all_geometries,"jaguar.xyz")
```

Python output

- Two functions, print and file.write()
  - print prints to standard output, appends new line

```
print "Hi There!"
```

  - file.write prints to file, does not automatically append a new line

```
file.write("Hi There!\n")
```

- Formatted output similar to C printf

```
file.write("%s has %d valence electrons\n" % ("C",4))
```

  - % operator puts the following tuple into the format characters

%s	String
%d	Integer (also %i)
%10.4f	Float 10 characters wide, with 4 decimal characters

write_xyz_file function

```
def write_xyz_file(all_geometries,filename):
  file = open(filename,'r')
  for geo in all_geometries:
       nat = len(geo)
       file.write('%d \n\n' % nat)
       for atom in geo:
         sym,x,y,z = atom
         file.write('%s %f %f %f\n' % (sym,x,y,z))
  return
```

Example: Jaguar output reader

- Three down:

```
#!/usr/bin/python
import os
energy_list = get_all_energies("jaguar.out")
all_geometries = get_all_geos("jaguar.out")
write_xyz_file(all_geometries,"jaguar.xyz")
plot_energy_values(energy_list)
os.system("xbs jaguar.xyz")
```

- Look at how plot data:

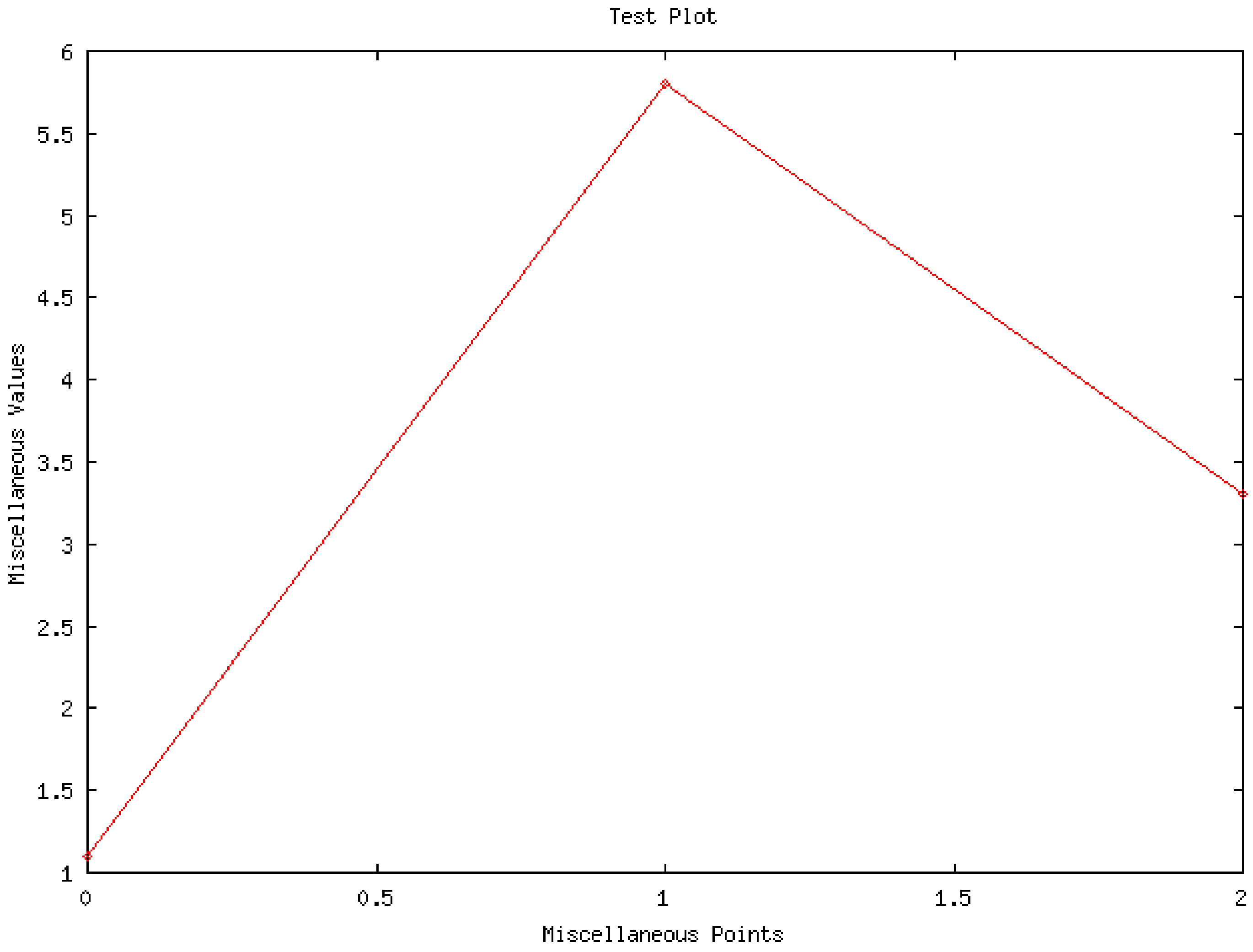
```
plot_energy_values(energy_list)
```

Gnuplot Module

- External module to handle plotting via gnuplot

```
import Gnuplot
g = Gnuplot.Gnuplot()
g.title('Test Plot')
g.xlabel('Miscellaneous Points')
g.ylabel('Miscellaneous Values')
g('set data style linespoints')
g.plot([[0,1.1],[1,5.8],[2,3.3]])
raw_input('Press return to continue...') #pause
```

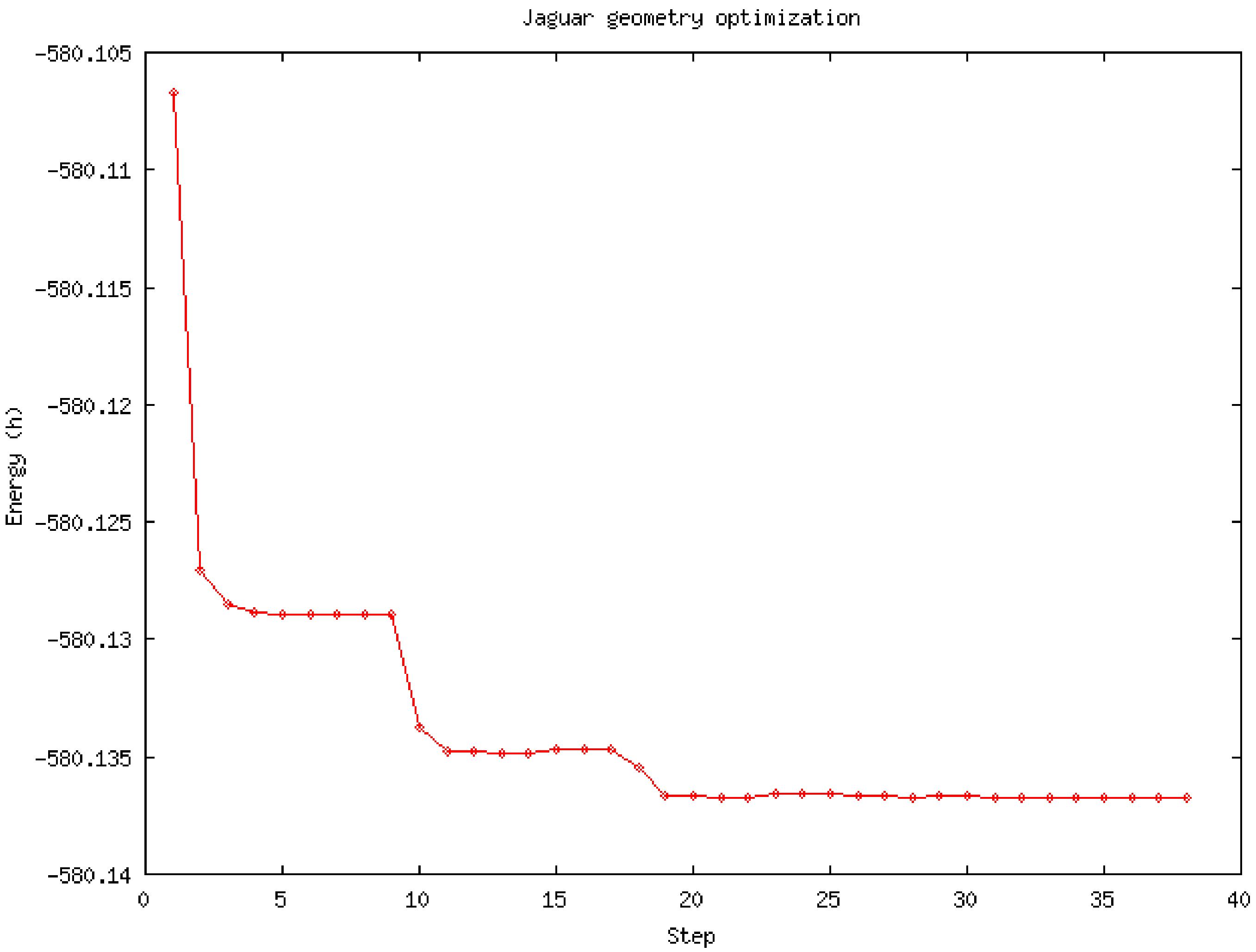
Gnuplot Sample Graph



Gnuplot Data

```
def plot_energy_values(energy_list):
  steps = range(len(energy_list))
  d = Gnuplot.Data(steps,energy_list)
  g = Gnuplot.Gnuplot()
  g.title('Jaguar optimization')
  g.xlabel('Step')
  g.ylabel('Energy (h)')
  g('set data style linespoints')
  g.plot(d)
  raw_input("Press any key to continue...")
  return
```

Output from Jaguar Optimization



Example: Jaguar output reader

- Three down:

```
#!/usr/bin/python
import os
energy_list = get_all_energies("jaguar.out")
all_geometries = get_all_geos("jaguar.out")
write_xyz_file(all_geometries,"jaguar.xyz")
plot_energy_values(energy_list)
os.system("xbs jaguar.xyz")
```

- Calling external functions:

```
os.system("xbs jaguar.xyz")
```

Calling external functions

- os.system(command) executes command in a shell

```
os.system("ls")
```

- Here we use the command to spawn an external viewer for the XYZ file:

```
os.system("xbs jaguar.xyz")
```

Importing and $PYTHONPATH

- Environmental variable PYTHONPATH
  - Search list of modules to import

```
% setenv PYTHONPATH .:/ul/rpm/python
```

- Import previously written modules:

```
from readers import xyzread
geo = xyzread("h2o.xyz")
for atom in geo:
  symbol, x, y, z = atom # break apart tuple
  print symbol, x, y, z
```

- or

```
import readers
geo = readers.xyzread("h2o.xyz")
for atom in geo:
  symbol, x, y, z = atom # break apart tuple
  print symbol, x, y, z
```

References

- Web Pages
  - http://www.python.org Python Web Site, lots of documentation
  - http://www.wag.caltech.edu/home/rpm/python_course/python_quick.html Python Quick Reference
- Books
  - *Learning Python*, Mark Lutz, David Ascher, Frank Wilson, ORA
  - *Programming Python*, Mark Lutz, ORA
  - *Python Programming on Win32*, Mark Hammond and Andy Robinson, ORA