**Classes continued.**

Here is another example. One needs a gff file in the same directory as the code. See below.

#This script reads in GFF file and processes it to make information callable using a gene name

class Gene(object):  
  
 def \_\_init\_\_ (self, gene\_info):  
 self.gene=gene\_info[8]  
 self.ch=gene\_info[0]  
 self.start=gene\_info[3]  
 self.end=gene\_info[4]  
 self.strand=gene\_info[6]def length(self): #calculates the length  
 length=(int(self.end) - int(self.start)+1)  
 return length  
  
 def make\_fasta\_head(self):  
 header= (">" + self.gene + " " + self.ch)  
 return header  
  
file\_obj=open("P\_out2") *#this is(parsed) gff information. I choose only lines with genes and selected 1000.*gene\_dict={} #create a dictionary to add key and values  
  
for line in file\_obj: *#captures one line at a time.  
 # Chr01 phytozomev11 gene 706 6715 . - . ID=Phvul.001G000400.v2.1;Name=Phvul.001G000400;ancestorIdentifier=Phvul.001G000400.v1.0* temp\_list = line.rstrip("\n").split("\t") *#generates a list* name\_temp=temp\_list[-1].split(";") *#generates a list from the last array element* name= (name\_temp[1])[5:len(name\_temp[1])] *#removes Name= to enable Phvul.001G000400 format* temp\_list[8]= name *#replace the last element of the array* gene\_dict[name]=Gene(temp\_list) *#make an object, creating instances of the class as a value of gene\_dict dictionary*for key in gene\_dict.keys ():  
 print (gene\_dict[key].make\_fasta\_head())  
 print (str(gene\_dict[key].length()))

**Sets**

The Python set type is like a dictionary that does not store any values. The set just stores a set of keys and allows us to very rapidly check whether or not a particular key is in the set. Seeing if an element or multiple elements are in data is done fairly frequently. For example, we may have a list of differentially expressed genes and subset of these genes have certain Gene Ontology terms. We may have a list of genes differentially expressed in treatment #1 and in treatment #2 and want to know which ones overlap.

I made three files from a Phaseolus .gff file parsed to contain lines with genes only.

head -n 100 P\_out > P\_out4 # used as GO genes. Pretend these genes have a shared GO term.

head -n 1000 P\_out > P\_out2 # used as expt1 genes

head -n 1500 P\_out | tail -1000 > P\_out3 # used as expt2 genes

We can use some of the code above to get gene names into sets.

def get\_name(file,set):

f\_obj=open("file")

for line in f\_obj: *#captures one line at a time.* temp\_list = line.rstrip("\n").split("\t") *#generates a list* name\_temp=temp\_list[-1].split(";") *#generates a list from the last array element* name= (name\_temp[1])[5:len(name\_temp[1])] *#removes Name= to be Phvul.001G000400 format* temp\_list[8]= name *#replace the last element of the array*

set.add(name). #here we add to the set

return set

exp\_1=set() #define sets

exp\_2=set()

exp\_3=set()

e\_1\_genes=get\_name("P\_out2",exp\_1) #calling function with args

e\_2\_genes=get\_name("P\_out3",exp\_2)

go\_genes=get\_name("P\_out4",exp\_3)

There are a number of ways to investigate sets. There is a function for finding elements not shared between sets and many other methods. The len() function returns the number of elements in a set, and the “in” and “not in” operators can be used to test for membership.

print (e\_1\_genes)

print(**"**experiment one gene number is "+ str(len(e\_1\_genes)))

print ("Phvul.001G000400" in e\_1\_genes)

print ("Phvul.001G000400" not in e\_1\_genes)

Set union can be performed with the | operator. For |, both operands must be sets.

print (str(len(e\_1\_genes | e\_2\_genes))) #all the genes in e1 and e2

Set union can also be obtained with the “.union()” method. The method is invoked on one of the sets, and the other is passed as an argument. The “.union()” method will take variables such as lists as an argument. It converts it to a set and then performs the union.

#what are all the genes that changed?

print (str(len(e\_1\_genes.union(e\_2\_genes)))

Intersection can be done with “.intersection” or “&”

#what are the genes that changed in both treatments?

print (str(len(e\_1\_genes.intersection(e\_2\_genes))))

print (str(len(e\_1\_genes & e\_2\_genes)))

#what are the genes in treatment 1 that have GO terms?

print (str(len(e\_1\_genes.intersection(go\_genes))))  
#what are the genes in treatment 2 that have GO terms?

print (str(len(go\_genes.intersection(e\_2\_genes))))

Note that we can do intersects and unions using lists and dictionaries, but this method is easier and faster!