Classes Code Continued…

We can tie object creation and the assignment of object’s variables together, so it is not possible to create an object without assigning values. This constructor method’s name is \_\_ init \_\_ and its arguments are in parentheses. We would remove the set.variables method with this constructor method.

def \_\_init\_\_(self, sequence, gene\_name, species\_name): *#three underscores before and after init* self.sequence= sequence  
 self.gene\_name= gene\_name  
 self.species\_name=species\_name

d1= DNARecord("AAAATTTT","TGA","Zea mays")d2= DNARecord("AAAACTAATAG","TB1","Zea mays")  
for r in [d1, d2]:  
 print (r.gene\_name + " from " + r.species\_name + " is " + r.sequence, end="")  
 print (" " + str(r.get\_AT()) + " is AT frequency")  
 print ("... and complement is " + r.complement())

Now, if we try to make an object without the arguments, we get an error.

D3=DNARecord() #Error.

# TypeError: \_\_init\_\_() missing 3 required positional arguments: 'sequence', 'gene\_name', and 'species\_name'

Here, we created objects that create variables to hold data and applied methods to get the answers. This is not so different from what we do with (other) python objects such as string objects.

dna=("ATGGATGA")   
a=dna.count("A")  
t=dna.count("T")  
print ((a+t)/len(dna))

The string object and our object both store their own variables and calculate values of interest. Once we have developed a new class, it behaves like a built-in class e.g DNARecord object is like a File, String, or Integer object. We can pass it as an argument or store it in a list, for example.

Let’s do another example (based on Ekmekci et al. 2016).

class SimpleClass (object):  
 a = 1234 #a variable/member stored in an object  
 def f(self): #a method stored in an object  
 return "Hello, I am here!"  
  
simpleObject = SimpleClass()print(simpleObject.a)  
print(simpleObject.f())

Assigning an object’s variable to another variable works probably as you expect:

sc1 = SimpleClass() #sc1 is an object

sc2 = sc1 #sc2 is the same as sc1 - it points to sc1

sc3 = SimpleClass() #sc3 is separate from sc1

Another example using the constructor class:

class SC3 (object):

#Note the lack of any members/variables. See \_\_init\_\_.

def \_\_init\_\_(self, aVal):

self.a = aVal

self.b = "Hello, I am here in \_\_init\_\_"

def getA(self):

return self.a

def getB(self):

return self.b

def changeB(self, newVal):

self.b=newVal

sc = SC3(4321)

print(sc.getA())

print(sc.getB())

sc.changeB("Yes, Message was changed.")

print(sc.getB())