6.189 - Intro to Python
IAP 2008 - Class 6
Lead: Aseem Kishore

Lab 8: Working with Objects

Problem 1 – Quick exploration of member functions

We've taught you that objects encapsulate data, and they provide member functions that operate based on that data. Some member functions change that data – in that case, they're modifiers and the object is mutable – while others simply return new data.

The syntax for accessing member functions of objects is:

```
variable . member_function ( [any arguments required] )
```

where variable points to an object in the heap.

Since the primitive types int, float and bool are NOT objects, they don't support any member functions. So you CAN'T do 5.something() or True.something().

However, strings are objects, so they support member functions. Same with tuples, and same with lists. What member functions do each support? Python provides a "dir" function that takes a variable and lists the member functions that the object supports.

Try this:

```
a = "hello"
dir (a)

b = (1, 2, 3)
dir(b)

c = [1, 2, 3]
dir(c)
```

You should get out a bunch of what looks like jumbled nonsense. But those things are actually member functions of each object. You'll generally see two types of member functions:

__functionname__ functions are generally not meant to be explicitly called. Instead, they're used by Python to do other things.

Try this:

```
a = "A"
b = "B"

print a < b            ← "A" comes before "B" alphabetically, so this is True
print a.__lt__(b)
```

We see the same result. It turns out that whenever you write object1 < object2, Python looks for a __lt__ member function in a and calls it with b as the argument. (In actuality, there is a little more to the story, but we won't worry about that now.)

The __functionname__ style exists because it's meant to explicitly make it clear that this function is not meant to be called normally. It's a system function, so in general, we should never be calling functions with names like __functionname__.

The rest of the functions are all functions that you can use! For help with any, type:

```
help(variable.functionname)
```

Take a quick look at some of the functions for strings, lists and tuples... we'll be needing them.

Problem 2 – Genetic mutations

At the top of your file, define four snippets of "chromosomes":

```
A = "gtggcaacgtgc"
B = "gtagcagcgcgc"
C = "gcggcacagggt"
D = "gtgacaacgtgc"
```

Let's say there are two methods to find the "distance" of two chromosomes -- that is, how much the two have varied through mutations. One method is to go letter by letter and count the number of discrepancies. Another method is to sum the discrepancies of a's (e.g. chromosome x has 5 a's, chromosome y has 7 a's), c's, g's and t's.

Are the two methods the same? We don't know, let's find out.

Write a function that uses method 1. It will take two chromosomes, and go letter by letter in each, comparing them. It then returns the total number of discrepancies it finds.

Write another function that uses method 2. It will take two chromosomes, and return the sum of 'a' discrepancies, 'c' ones, and so on.

Hint: Strings have many useful member functions. Try to find some that will let you do method 2 easily, without having to use any loop or do any complicated math.

Then call each function on each of the combinations (A,B), (A,C), (A,D), (B,C), (B,D) and (C,D) and see which methods tell you which pair of chromosomes is "furthest" (i.e. most varied) and which pair is "closest" (i.e. least varied).

Remember to use dir()!

Important note: From this point on, we are going to stray from interactive programs that use raw_input and go to solving complex problems instead. You should not use a raw_input function anywhere in this program.