Genome 559
Intro to Statistical and Computational Genomics
2009

Lecture 15b: Classes and Objects, II

Larry Ruzzo

1 Minute Reflections

Your explanation of classes was much clearer than the book's!

I liked the explanation of objects and classes.

Class was fine today. The new programming stuff is a little hard but I am slowly getting it.

Classes and objects will require some practice for me to feel comfortable with them.

Until now Python seemed very intuitive. Now with class definitions (esp. the use of "self" and __init__) Python became counter-intuitive. But your explanations were clear. Thanks for taking the time to explain slowly.

Today's explanations were good, but I still couldn't figure out the practice problems.

Today's class was OK. But the class and object part was a bit confusing to me. I guess I just need more practice.

Where I previously felt confident with the programing, I now feel a bit overwhelmed and a bit lost.

It would be helpful for me to have a more generalized example of the new code elements we're talking about to know what is essential for Python and what is optional. A good place for this would be a summary slide of important functions/code.

> *Will try.*

What are the general limits for classes you create? Are classes something "risky" to create? (I.e., could you really screw up Python by creating a class with a really huge error in it?)

> *Classes are optional, and no, not risky. About the worst you can do is inadvertently duplicate/override a name that Python expects and uses.*

I think it would be useful to see some examples of larger Python programs and go over what each part does during class.

Yes, I plan to do this.

If you are still thinking about what to do with the rest of the class, it would be nice to have a lot of practice time with Python, with problems that incorporate the things we've learned over the last quarter.

I'm definitely open to suggestions and this is a good one.

Today

A comment on old homework

More on the practice problems from Thursday

More fun with classes

Comment on old homework

It's important to read/define the problem carefully

Programs that do something slightly different are a bad idea

Example:

- In a PHYLIP file, the first 10 characters are the species name
- Commonly, the name ends with a space and has no spaces inside
- If you rely on this, easy to use string.split()
- BUT legal PHYLIP files can break that code:
    - `Homosapie ATGCGGCGGTCAATC` - OK
    - `HomosapienATGCGGCGGTCAATC` - FAIL
    - `Homo sapieATGCGGCGGTCAATC` - FAIL
- In this case, using slice[:10] is better

If you have a format specification, use it: don't just look at the sample file

(If the two don't match, talk to the person who asked you to write this program: they may be confused, and the sooner you find out, the better.)

(And yes, my GenBank homework was perhaps teaching you bad habits...)

More on Classes

Much in modern programming languages is motivated by the need to write large programs

- BioPython is 25 megabytes, ~0.5 million lines
- And that isn't near the large end of the scale
- Large programs aren't just small programs on steroids
- (Not always easy to appreciate until it's too late)

Python modules are one such feature

Classes/"object oriented programming" are another

My goal is *not* to make you instant experts at this, but to acquaint you with the issues so you can use "object-oriented" tools, e.g., BioPython, and won't be intimidated by these features.

What’s in a Name?

Management of (many!) names is one issue

```
myseq = file.readline()
frags = digest(mysequence)
```

Hmm, did you mean:

EcoR1 + DNA? `frag = dna_digest(myseq)`

trypsin + protein? `frag = tryp_digest(myseq)`

Oh, and your pal sent you `rev_comp_DNA()`

Will you ever forget/use the wrong name/case?

Modules Might Help

Have a module named `DNA` for all your DNA-based tools

```
import DNA
antisense = DNA.rev_comp(myseq1)
frags = DNA.digest(myseq1)
```

Another module named `prot` for protein tools

```
import prot
frags = prot.digest(myseq2)
```

At least you now have consistent spelling

But you might still twitch and call the wrong `.digest()`

“Classes” might help?

Have separate classes for protein vs DNA sequences, each with appropriate methods

```
class SeqDNA:
  def digest(theseq): ...
  def rev_comp(aseq): ...
class SeqProt:
  def digest(someseq): ...
myseq = SeqDNA(file.readline())
frags = SeqDNA.digest(myseq)
```

yes, this really works

A lot like the “module” version: consistent spelling, but still error-prone, *and* extra “constructor” step

Classes help more: methods & the “self” shorthand

Instead of:

$$\begin{tikzcd}\texttt{classname.methodname(class\_instance)} \\ \texttt{class\_instance.methodname()} \arrow[u, "\boxed{\text{Automatically converted}}"']\end{tikzcd}$$

Do this:

E.g.:

$\texttt{myseq.digest()} \xrightarrow{\boxed{\text{Auto conv}}} \texttt{SeqDNA.digest(myseq)}$

How? The class instance knows what class it’s in, and effectively “inherits” that class’s methods.

Classes help more

Have separate classes for protein vs DNA sequences, each with appropriate methods

```
class SeqDNA:
  def digest(self): ...
  def rev_comp(self): ...
class SeqProt:
  def digest(self): ...
myseq = SeqDNA(file.readline())
frags = myseq.digest()
```

Better than the "module" version: yes, still the extra "constructor" step, but since objects know which class they're in, you *always* get the class-specific method

Continuing "Date" example

```
  class date:
   def __init__(self, day, month) :
     self.myday = day
     self.mymonth = month
   def printdate(self)
     print self.myday, self.mymonth

mydate = date(15,"January")
mydate.printdate()
15 January
```

Practice

Write a function for our date class that allows us to add a number to a date

Algorithm:

add the number to the day; if this goes past the end of a month, advance to the next month; repeat

Step 1: Set up a dictionary mapping month name (key) to number of days in month (value)

Step 2: Write a function nextmonth(month_name) returning name of the next month.

E.g., nextmonth(“Apr”) == “May”, nextmonth(“Dec") == “Jan"

You can do this with a big if statement, but there are easier ways

(Hint: make a list of months with an extra “January" at the end)

Step 3: Copy the class definition into your program file

Add a new class function add(self, numdays)

This function accepts only positive number arguments

It should use the dictionary to find the number of days in a month, and the nextmonth function to find the next month