Genome 559
Intro to Statistical and Computational Genomics

Lecture 16b:
Classes and Objects, Part III
Larry Ruzzo

Outline

Printing more naturally

“Operator Overloading”

Inheritance

Shallow copy/deep copy

Printing Objects

Why is "print" fine for numbers, tuples, etc.

```
>>> print ("Jan",5)
('Jan', 5)
```

but funky for class instances?

```
print mydate
<__main__.date instance at 0x247468>
```

Yes, `mydate.printUS()` works, but seems clunky

A better way to print objects

Actually, "print" doesn't have special knowledge of how to print numbers, strings, tuples, ...

It just knows how to print strings, and relies on each *class* to have a `__str__()` method that returns a string representing the object.

"<__main__.date instance at 0x247468>" is the result of calling the *default* `__str__()` method.

You can write your own, tailored `__str__()` method to give prettier/more useful results

Printing dates

```
class Date(object):
  def __init__(self, day, month) :
    self.day = day
    self.mon = month

  def __str__(self) :
    return '%s %s'%(self.mon, self.day)

  add(self, numdays) :
    (etc., as before)

birthday = Date(3,"Sep")
print "It's ", birthday, ". Happy Birthday!"
```

Advanced topic: Allowing the plus sign

Similarly, how come "+" works (but differently) for numbers and strings and tuples and ..., but not for dates?

Yes, this works:

```
"party = mybirthday.addnew(4)"
```

to add numbers to dates, but this:

```
"party = mybirthday + 4"
```

seems so much more natural. Can we do it?

Advanced topic: Overloading "+"

Yes! Again, '+' isn't as smart as you thought; it calls class-specific "add" methods ("__add__( )") to do the real work:

```
def __add__(self, numdays) :
  newmon = self.mon
  newday = self.day + numdays
  while newday > daysinmonth[newmon] :
    newday = newday - daysinmonth[newmon]
    newmonth = nextmonth(newmon)
  return Date(newday,newmon)

# usage example
mybirthday = Date(6,"Jul")
party = mybirthday + 4
print mybirthday, party
```

`mybirthday.__add__(4)`

`Jul 6 Jul 10`

Operator overloading

This shows some of the power of classes in Python; we can make new classes, like Date, behave like built-in ones

Operator overloads involve names with underscores
Common operator overloading methods

```
__init__ # object creation
__add__  # addition (+)
__mul__  # multiplication (*)
__sub__  # subtraction (-)
__lt__   # less than (<)
__str__  # printing
__call__ # function calls
...      # And more: indexing, slicing, iteration, ...
```

Try "`>>>dir(object)`" in Python to see what's there

Pros and Cons

Good aspects of operator overloading

- Can make classes easier to use
- Uniformity: use your own classes just like built-in ones

Bad aspects:

- Might obfuscate things (overload the + sign to do subtraction...)
- The "implicit" function calls can be confusing to follow at first

Net: an advanced technique you may or may not need

Exceptions: almost all classes will need __init__ () functions, and __str__() is usually a good idea, too

Inheritance: do the common parts once

```
class Seq(object):
  def print_FASTA(self): ...
class DNA(Seq):
  def digest(self): ...
  def rev_comp(self): ...
class Prot(Seq):
  def digest(self): ...

myseq = DNA(file.readline())
frags = myseq.digest()
myseq.print_FASTA()
```

Superclass for seqs in general, with appropriate methods common to all

Separate *subclasses* for protein vs DNA sequences, with methods appropriate to each

myseq is a "DNA" object; doesn't have a "print_FASTA" method, but *inherits* it from Seq superclass

A key OO feature; done well, saves much work (done poorly–can be very confusing)

“Classes”: Summary

Most useful in (but not restricted to) large programs

Classes package together related data plus the functions (“methods”) appropriate thereto

Method calls automatically find the def of the given name within their own class, not some other one spelled the same

The relevant object is always passed to the method as its 1st parameter, called “self” by convention

Method names starting & ending with “__” are special, allowing “operator overloading” and other emulation of “standard” behavior

Feedback

Please go to WebQ link next to HW6 on class home page & fill out.

Practice

Definitely try dir(x) for x= some string, some int, some list, your Date class

More play with Date example, esp. add __str__

Try Seq/DNA/Prot example from 3 slides back.

- for prot.digest, just split on "W", say; DNA, on "AAA"
- What should digest return?
- do rev, maybe rev_compl