More on Classes, Biopython

Genome 559: Introduction to Statistical and Computational Genomics

Elhanan Borenstein
A quick review

- Class inheritance

```python
class HotDate(Date):
    __init__(self, day, month, year, toothbrush):
        super(day, month, year)
        self.bringToothbrush = toothbrush
```

- Exception Handling

```python
try:
    self.day = int(day)
except ValueError:
    print 'Date constructor: day must be an int'
```

- Creating your own Exception

  - Just inherit: `exceptions.Exception`
More about classes and inheritance
Consider two classes

class A:
    def __init__(self, number):
        self.num = number

Class B:
    str = "hello"
Relationships between classes

- There are two basic methods through which class B can “use” class A (e.g., have access to members of A):

  1. Class A has a member object of class B

     ```python
     Class B:
     str = "hello"
     obj_A = A(7)
     ```

  2. Class A inherits class B

     ```python
     Class B(A):
     str = "hello"
     ```

- How do we know when to use each of these methods?
The “has” vs. “is” test

- If B “has” an A:
  
  ```python
  Class B:
  str = "hello"
  obj_A = A(7)
  ```

- If B “is” (or “is a type of”) A:
  
  ```python
  Class B(A):
  str = "hello"
  ```

- Examples:
  - B describes a protein, A described a domain
  - B describes an enzyme, A described a protein
class Date:
    def __init__(self, day, month):
        self.day = day
        self.mon = month

    def printNice(self):
        print self.mon, "/", self.day

class Person:
    def __init__(self, name, DOB):
        self.name = name
        self.DOB = DOB

    def printNice(self):
        print "Name:", self.name
        print "DOB:",
        self.DOB.printNice()

person_1 = Person("John", Date(22, 11))
person_1.printNice()

Name: John
DOB: 11 / 22
Example 2

class Date:
    < AS BEFORE >

class Person:
    < AS BEFORE >

class Student(Person):
    def __init__(self, name, DOB, student_id):
        self.ID = student_id
        Person.__init__(self,name,DOB)

    def printNice(self):
        Person.printNice(self)
        print "ID: ", self.ID

student_1 = Student("John", Date(22, 11),32353)
student_1.printNice()

Name: John
DOB: 11 / 22
ID: 32353
Multiple inheritance

- A class can inherit from more than one class ...

```python
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    ...
    ...
    ...
    <statement-N>
```

- A good way to create a very powerful class by inheriting multiple capabilities
Beware of diamonds ...

```python
class Person:
    < AS BEFORE >

class Student(Person):
    < AS BEFORE >

class TA(Person):
    < ADDING COURSES AND ASSIGNMENTS >

Class StudentTA(Student, TA)
    < COOL?>
```

- This should work, right?
- What’s interesting about this case?
A very (very very) short introduction to Biopython
Biopython

- Biopython is a tool kit, not a program – a set of Python modules useful in bioinformatics

- Features include:
  - Sequence class (can transcribe, translate, invert, etc)
  - Parsing files in different database formats
  - Interfaces to progs/DBs like Blast, Entrez, PubMed
  - Code for handling alignments of sequences
  - Clustering algorithms, etc, etc.

- Useful tutorials at http://biopython.org
Making Biopython run on your computer

- Runs on Windows, MaxOSX, and Linux

- Go to http://biopython.org/
  - Look for download/install instructions
  - May require “Admin” privileges
Example: sequence class

- Hold the sequence string and an associated alphabet

```python
>>> from Bio.Seq import Seq  # seq class
>>> myseq = Seq("AGTACACTGGT")
>>> myseq.alphabet
Alphabet()
>>> myseq.tostring()
'AGTACACTGGT'
```
Example: sequence class, cont’

- More functionality than a plain string

```python
>>> myseq
Seq('AGTACACTGGT', Alphabet())

>>> myseq.complement()
Seq('TCATGTGACCA', Alphabet())

>>> myseq.reverse_complement()
Seq('ACCAGTGTACT', Alphabet())
```
Biopython and Blast

- Biopython can run Blast!
- Either locally or over net
- Save results
- Parse and analyze results
http://www.biopython.org

(get used to reading software documentation)
Sample problem #1

- In addition to the class Date you implemented last week, implement the following classes:
  - Time() – this class should maintain information about the time of the day (hour and minutes)
  - Meeting() – this class will be used to handle a meeting time slot (date, start time and end time).
  - Create an object of the class meeting (providing date, start and end time), and call its print method.
- Note: What should be the relationships between these 3 classes?
class Date:
    def __init__(self, day, month):
        self.day = day
        self.month = month
    def __str__(self):
        return '%s' % self.day+'/'+'%s' % self.month

class Time:
    def __init__(self, hour, minutes):
        self.H = hour
        self.M = minutes
    def __str__(self):
        return '%s' % self.H+':'+'%s' % self.M

class Meeting:
    def __init__(self, m_date, m_start, m_end):
        self.date = m_date
        self.start = m_start
        self.end = m_end
    def printNice(self):
        print "Meeting on", self.date, "from", self.start, "to", self.end

my_class = Meeting(Date(3,3), Time(3,30), Time(4,50))
my_class.printNice()
Sample problem #2

- Now, implement the class GroupMeeting that will be used to handle meetings of group of people. In addition to the details required for a Meeting class, this class should also store (and initialize and print) the names of the people that are to attend the meeting.
class Date:
    < AS BEFORE>

class Time:
    < AS BEFORE>

class Meeting:
    < AS BEFORE>

class GroupMeeting(Meeting):
    def __init__(self, m_date, m_start, m_end, people_list):
        Meeting.__init__(self, m_date, m_start, m_end)
        self.group = people_list
    def printNice(self):
        Meeting.printNice(self)
        print "The following people should attend:", self.group

g_meeting = GroupMeeting(Date(3,3),Time(3,30),Time(4,50),["Elhanan","Jim"])
g_meeting.printNice()

Meeting on 3/3 from 3:30 to 4:50
The following people should attend: ["Elhanan","Jim"]
Challenge Problem

1. Think which classes you would implement to model a network. What data should they hold? What methods should they provide.

2. Implement these classes and use them to model a simple 10-node network.

3. Use your network model to plot the degree of each node in the network.