

Welcome to
Python 2
Session #3

Michael Purcaro & The GSBS Bootstrappers

February 2014

michael.purcaro@umassmed.edu

Simple Book Class

```
class Book():  
    def __init__(self, title, author):  
        self.title = title  
        self.author = author  
  
joeBook = Book("Joe", "Joe's Story")  
daveBook = Book("Dave", "Dave's Tale")
```

Class Book

self.author

self.title

Definition

```
class Book():  
    def __init__(self, title, author):  
        self.title = title  
        self.author = author
```

Class Book

joeBook

self.author = "Joe"

self.title = "Joe's Story"

Usage 1

```
joeBook = Book("Joe", "Joe's Story")
```

Class Book

daveBook

self.author = "Dave"

self.title = "Dave's Tale"

Usage 2

```
daveBook = Book("Dave", "Dave's Tale")
```

Bed Record Class

```
1 class bedRecord():
2     def __init__(self, chrom, start, stop):
3         self.chr = chrom
4         self.start = start
5         self.stop = stop
6
7     def getLen(self):
8         return self.stop - self.start
9
10    def doesRecordOverlap(self, otherRecord):
11        return (self.start >= otherRecord.start and self.start < otherRecord.stop)\
12            or (self.stop >= otherRecord.start and self.stop < otherRecord.stop)
13
14    bed1 = bedRecord("chr7", 100050, 100079)
15    bed2 = bedRecord("chr7", 100025, 100060)
16    bed3 = bedRecord("chr7", 100090, 100110)
17
18    print bed1.getLen()
19
20    print bed1.doesRecordOverlap(bed2)
21    print bed1.doesRecordOverlap(bed3)
22
```

Bed Record Class

```
1 class bedRecord():
2     def __init__(self, chrom, start, stop):
3         self.chr = chrom
4         self.start = start
5         self.stop = stop
6
7     def getLen(self):
8         return self.stop - self.start
9
10    def doesRecordOverlap(self, otherRecord):
11        return (self.start >= otherRecord.start and self.start < otherRecord.stop) \
12            or (self.stop >= otherRecord.start and self.stop < otherRecord.stop)
13
14    bed1 = bedRecord("chr7", 100050, 100079)
15    bed2 = bedRecord("chr7", 100025, 100060)
16    bed3 = bedRecord("chr7", 100090, 100110)
17
18    print bed1.getLen()
19
20    print bed1.doesRecordOverlap(bed2)
21    print bed1.doesRecordOverlap(bed3)
22
```

The diagram illustrates the relationship between the `bedRecord` class and its instances. Purple arrows show the class methods `__init__`, `getLen`, and `doesRecordOverlap` being called from the class definition. Orange arrows show the instance methods `getLen` and `doesRecordOverlap` being called from the instance definitions. A green arrow shows the instance method `getLen` being called from the instance definition.

Class Member Functions

```
class bedRecord
```

```
bed1
```

```
    self.chr = "chr7"
```

```
    self.start = 100050
```

```
    self.stop = 100079
```

```
14 bed1 = bedRecord("chr7", 100050, 100079)
```

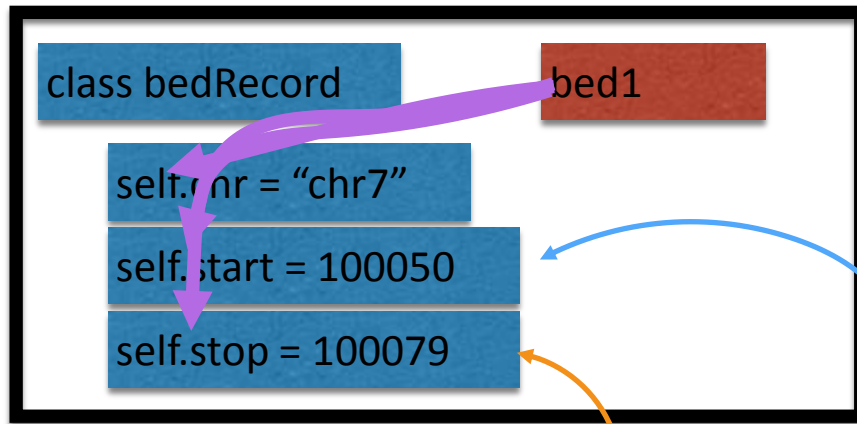
```
7  def getLen(self):  
8      return self.stop - self.start
```

Definition

```
18 print bed1.getLen()  
19
```

usage

Class Member Functions



```
14 bed1 = bedRecord("chr7", 100050, 100079)
```

```
7 def getLen(self):  
8   return self.stop - self.start
```

Definition

```
18 print bed1.getLen()  
19
```

usage

Don't need to use self when
using the member function

class bedRecord

bed1

self.chr = "chr7"

self.start = 100050

self.stop = 100079

class bedRecord

bed2

self.chr = "chr7"

self.start = 100025

self.stop = 100060

```
10 def doesRecordOverlap(self, otherRecord):
11     return (self.start >= otherRecord.start and self.start < otherRecord.stop)\
12     or (self.stop >= otherRecord.start and self.stop < otherRecord.stop)
13
```

```
20 print bed1.doesRecordOverlap(bed2)
```


class bedRecord

bed1

self.chr = "chr7"

self.start = 100050

self.stop = 100079

class bedRecord

bed2

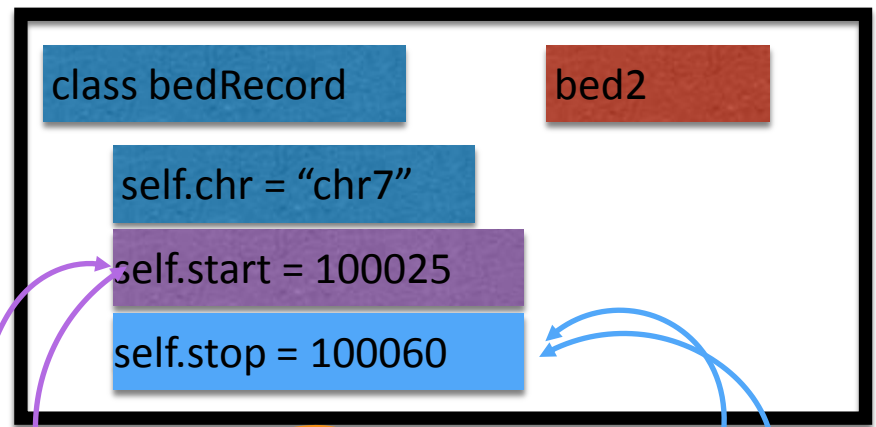
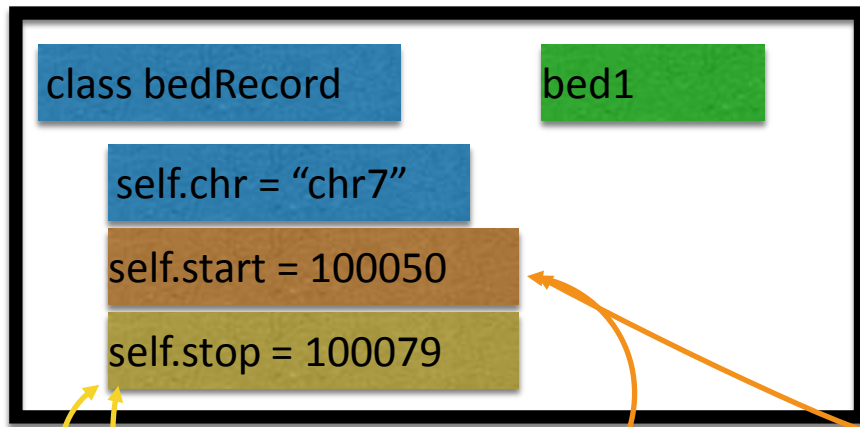
self.chr = "chr7"

self.start = 100025

self.stop = 100060

```
10 def doesRecordOverlap(self, otherRecord):  
11     return (self.start >= otherRecord.start and self.start < otherRecord.stop) \  
12     or (self.stop >= otherRecord.start and self.stop < otherRecord.stop)  
13
```

```
20 print bed1.doesRecordOverlap(bed2)
```



```
10 def doesRecordOverlap(self, otherRecord):
11     return (self.start >= otherRecord.start and self.start < otherRecord.stop) \
12     or (self.stop >= otherRecord.start and self.stop < otherRecord.stop)
13
```

```
20 print bed1.doesRecordOverlap(bed2)
```

Extended Exercise 4

- Rework your code from the previous Extended Exercises into a ChIPseqData class, similar to

```
class ChIPseqData:
```

```
    def __init__(self, paths, url):
```

- The class initializer should only download the chipseq data if needed
 - What are the pros/cons of downloading the data during class initialization?
- The class should have a function to return the number of peaks found in a given chromosome
- The class should also have a function to compute the percentage of a given chromosome covered by peaks
- Would a Peak class (that understood how to parse each line of the narrow peak) be useful? Try it!

```
class Paths():
    def __init__(self, lectureNumber):
        self.homeFolder = os.path.abspath(os.path.expanduser("~/"))
        self.desktopFolder = os.path.join(self.homeFolder, "Desktop")
        self.python2folder = os.path.join(self.desktopFolder, "python_2")
        lecture = "lecture_" + str(lectureNumber)
        self.lectureFolder = os.path.join(self.python2folder, lecture)
        print "today's lecture folder location will be:", self.lectureFolder
        Utils.mkdir_p(self.lectureFolder)

    def makeFilePath(self, fn):
        return os.path.join(self.lectureFolder, fn)
```

```

class ChipseqData:
    def __init__(self, paths, url):
        self.paths = paths
        self.url = url
        self.fnp = Utils.get_file_if_size_diff(self.url, paths.lectureFolder)

    def getPeaks(self, chr):
        peaks = []
        with open(self.fnp) as f:
            for line in f:
                toks = line.split()
                if chr != toks[0]:
                    continue
                peaks.append(toks)
        return peaks

    def numPeaks(self, chr):
        return len(self.getPeaks(chr))

    def computePercentageChromosomeCovered(self, chr, chromosomeLength):
        peaks = self.getPeaks(chr)
        numBases = 0
        for toks in peaks:
            numBases += int(toks[2]) - int(toks[1])
        return "{0:.2f}%".format(float(numBases) / chromosomeLength * 100)

```

<http://bioinfo.umassmed.edu/bootstrapers/bootstrappers-courses/python2/lecture2/lecture2.txt>

```
paths = Paths(3)
```

```
chipData = ChIPseqData(paths,  
"http://bib3.umassmed.edu/~purcarom/Python2/Lecture1/ENCFF002C  
OQ.narrowPeak")
```

```
print "number of peaks on chromosome 7:",  
chipData.numPeaks("chr7")
```

```
print "% of chromosome 7 covered by peaks:",  
chipData.computePercentageChromosomeCovered("chr7",  
159138663)
```

Extended Exercise 5

- Build a class “HG19chrSize” that holds chromosome lengths
 - Have its constructor download and parse the file
 - Store chromosome lengths in a dictionary
 - Key of “chromosome name” (i.e. “chr6” or “chrX”)
 - Value of integer
 - Have a class method “length(chrmosomeNum)” that returns the length of the chromosome requested
- Chromosome lengths in:

<http://bioinfo.umassmed.edu/bootstrappers/bootstrappers-courses/python2/lecture1/hg19.chrom.sizes>

Extended Exercise 6

- **Goal:** Run `numPeak()` and `computePercentageChromosomeCovered()` for chromosomes 1 to 22
 - Store the `numPeak()` output into a dict:
 - `numPeaks = {}`
 - `numPeaks["chr1"] = ...`
 - Store the `computePercentageChromosomeCovered()` output in
 - `percChromCovered = {}`

Extended Exercise 7

- Change the ChipseqData class to use Peak class:

```
class Peak:
    def __init__(self, line):
        toks = line.split()
        self.chr = toks[0]
        self.start = int(toks[1])
        self.end = int(toks[2])

    def length(self):
        return self.end - self.start
```

Static Class Methods

- Methods that live in a class, but don't need any access to data (via `self`) in that class
- One use: organize miscellaneous functions together

class **Utils**:

@staticmethod

def **mkdir_p**(path):

...

@staticmethod

def **get_file_if_size_diff**(url, path):

Called without using an object!

`Utils.mkdir_p(path)`

`Utils.get_file_if_size_diff(url, path)`

Building Blocks: list comprehension

- Quick way to build certain kinds of lists

```
a = [x for x in range(10)]  
print a
```

```
b = [x*x for x in range(10)]  
print b
```

```
c = [str(x) for x in range(10)]  
print c
```

```
q = 5  
d = [x+q for x in range(10)]  
print d
```

Building Blocks: list comprehension

```
fnp = paths.makeFilePath("ENCFF002COQ.narrowPeak")

with open(fnp) as f:
    allPeaks = [Peak(x) for x in f]

print len(allPeaks)
```

Building Blocks: map

- Another way to build a list
- Applies a function to every element in a list

```
a = map(lambda x: x, range(10))  
print a
```

```
b = map(lambda x: x*x, range(10))  
print b
```

```
c = map(str, range(10))  
print c
```

```
q = 5  
d = map(lambda x: x+q, range(10))  
print d
```

Building Blocks: map

- Another way to build a list
- Applies a function to every element in a list

```
a = map(lambda x: x, range(10))  
print a
```

“anonymous” function!
→ Function with no name

```
b = map(lambda x: x*x, range(10))  
print b
```

```
c = map(str, range(10))  
print c
```

For examples on this slide, map
assumes the function takes only
1 argument

```
q = 5  
d = map(lambda x: x+q, range(10))  
print d
```

Building Blocks: map

```
fnp = paths.makeFilePath("ENCFF002COQ.narrowPeak")

with open(fnp) as f:
    allPeaks = map(Peak, f)

print len(allPeaks)
```

Homework #1

- Work through these problems from <http://rosalind.info/problems/list-view/>
- DNA Counting DNA Nucleotides
- RNA Transcribing DNA into RNA
- REVC Complementing a Strand of DNA
- GC Computing GC Content
- HAMM Counting Point Mutations
- SPLC RNA Splicing
- PROT Translating RNA into Protein
- SUBS Finding a Motif in DNA
- PRTM Calculating Protein Mass
- REVP Locating Restriction Sites

Review on February 12, 2015