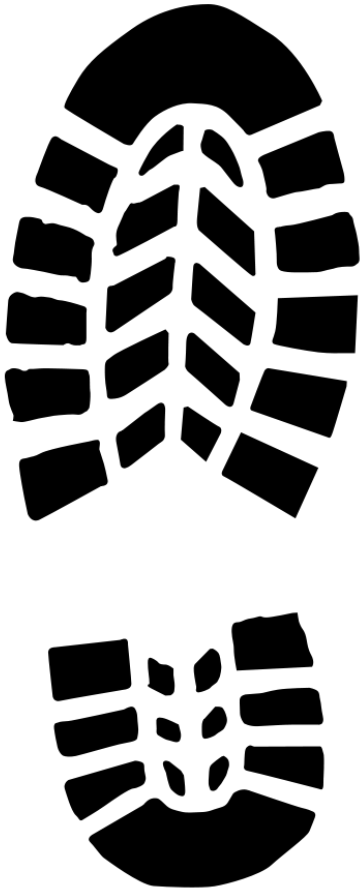

Welcome to

Python 2

Session #2

Michael Purcaro & The GSBS Bootstrappers

February 2014

michael.purcaro@umassmed.edu

Extended Exercise 1

Goal: count how many signal peaks are present in processed ENCODE ChIP-seq data on chromosome 7

url:
http://bib3.umassmed.edu/~purcarom/Python2/Lecture1/ENCFF002COQ.narrowPeak

File format:

genome.ucsc.edu/FAQ/FAQformat.html#format12

Answer hint: between 2000 and 3000

Extended Exercise 2

Modify code from Extended Exercise 1 to compute what percentage of chromosome 7 (assume hg19) is covered by peaks.

Length of chr7 in hg19: 159138663

(Length of HG19 chromosomes in hg19.chrom.sizes in bioinfo.umassmed.edu/bootstrappers/bootstrappers-courses/python2/lecture1/)

Answer hint: <5%

Building Blocks: Functions

- Functions help divide code up into smaller chunks
  - easier to understand
  - easier to test individual function
  - encourages code reuse
  - can hide details not pertinent to higher-level understanding of code
  - helps introduces structure to code
    - in some ways analogous to English paragraphs
      - i.e. more difficult to read a paper or novel if there were no paragraph breaks or sentences: all the words just run together

Building Blocks: Functions

```
def functionName(parameters):
    code (no more than a "screenful",
              i.e. approx. <40 lines)
    optionally, return something

def chrNumAsString(num):
    return "chr" + str(num)
```

Download a file if size changed

```
def get_file_if_size_diff(url, d):
    fn = url.split('/')[-1]
    out_fnp = os.path.join(d, fn)
    net_file_size = int(urllib.urlopen(url).info()['Content-Length'])
    if os.path.exists(out_fnp):
        fn_size = os.path.getsize(out_fnp)
        if fn_size == net_file_size:
            print "skipping download of", fn
            return out_fnp
        else:
            print "files sizes differed:"
            print "\t", "on disk:", fn_size
            print "\t", "from net:", net_file_size
    print "retrieving", fn
    urllib.urlretrieve(url, out_fnp)
    return out_fnp
```

Exercise 3

- Convert your code from Extended Exercises 1 and 2 into functions!
- Hints:
  - Make a function to
    - Download a url to a certain file
    - Return a list of peaks for a given chromosome
    - Compute the percentage of a given chromosome covered by the peaks

Building Blocks: dictionary

- Easy way to associate one thing (called a *key*) to another (called the *value*)
- Example: simple daily calendar

```
dailyCalendar = {}
dailyCalendar = { "9AM" : "PI Meeting",
                  "11AM" : "conference ",
                  "4PM" : "book club"
                      }

print dailyCalendar["4PM"]
dailyCalendar["4PM"] = "PI meeting"
print dailyCalendar["4PM"]
print dailyCalendar["3PM"]
```

Building Blocks: dictionary

```
dailyCalendar = { "9AM" : "PI Meeting",
                  "11AM" : "conference ",
                  "4PM" : "book club"
                    }

if "3PM" in dailyCalendar:
    print dailyCalendar["3PM"]
```

Building Blocks: dictionary

```
dailyCalendar = { "9AM" : "PI Meeting",
                  "11AM" : "conference ",
                  "4PM" : "book club"
                    }

for time, obligation in dailyCalendar.iteritems():
    print "at", time, ":\t", obligation
```

Building Blocks: dictionary

- Dictionary can hold wide variety of data

```
d = {}
d[1] = ["a", "b", ["q", "r", "s"]]
d[2] = 123
d[3] = set([1,1,2,3,5,8,13])

print d[2]
123
```

Building Blocks: set

- Keeps only unique items

```
nums = set()

nums.add(1)
nums.add(1)
nums.add(2)
nums.add(3)
nums.add(5)
nums.add(8)
nums.add(8)

print nums

print 10 in nums
```

Building Blocks: defaultdict

- Like a normal dictionary, but supplies a default value if the key is not in the dictionary

```
from collections import defaultdict

dailyCalendar = defaultdict(str)
dailyCalendar["9AM"] = "PI Meeting"
dailyCalendar["11AM"] = "conference call"
dailyCalendar["4PM"] = "book club"

print dailyCalendar["4PM"]
print dailyCalendar["3PM"]
```

Extended Exercise 3

- Goal: Count how many DNase-seq peaks overlap ChIP-seq peaks on chromosome 7

- ChIP-seq data url (same as before):
  http://bib3.umassmed.edu/~purcarom/Python2/Lecture1/ENCFF002COQ.narrowPeak
- DNase-seq data url:
  http://bib3.umassmed.edu/~purcarom/Python2/Lecture2/ENCFF001VZW.narrowPeak

- Hint: use a set

- Answer hint: between 1000 and 2000

Building Blocks: Classes

- In an ideal world, a class contains
  - data (numbers, strings, lists, other classes, etc.)
  - the functions allowed to manipulate the data
- Provides further mechanisms to give code structure
  - Allows data and concepts to be encapsulated (i.e. hidden or only occur in one place)

Building Blocks: Classes

- Classes have constructors or initializers that perform certain operations (like setting up data structures, etc.) when the class is first created
- Python classes have a "`self`" variable that understands how to access data in the class

Building Blocks: Classes

```
class ExampleKlass():
    def __init__(self, strData):
        self.data = strData

    def printData(self):
        print self.data

ek = ExampleKlass("hi!")
ek.printData()
```

Classes and Objects

- In object-oriented programming, a “class” is the code that defines
  - the class variables
  - the functions allowed to operate on those class variables
  - No memory taken up
- An “object” is an instance of a class
  - Memory will be allocated for the variables in the object

Class Class Exercise

Make a “Paths” class that encapsulates the path manipulations we performed earlier

```
homeFolder = os.path.abspath(os.path.expanduser("~"))
desktopFolder = os.path.join(homeFolder, "Desktop")
python2folder = os.path.join(desktopFolder, "python_2")
lecture1folder = os.path.join(python2folder, "lecture_1")
print "today's lecture folder location will be:", lecture1folder
mkdir_p(lecture1folder)
```

Why use classes?

- What data needed to describe a book?

```
title = ""
author_first_name = ""
author_last_name = ""
year_published = ""
num_pages = ""
```

Why use classes?

- What data needed to describe a book?

```
title = ""
author_1_first_name = ""
author_1_last_name = ""
author_2_first_name = ""
author_2_last_name = ""
year_published = ""
num_pages = ""
```

Why use classes?

- What data needed to describe a book?

title = ""
author_1_first_name = ""
author_1_last_name = ""
author_2_first_name = ""
author_2_last_name = "“
...
author_n_first_name = “”
author_n_last_name = “”
year_published = ""
num_pages = "“
front_cover_picture = “”
back_cover_picture = “”
language = “”
edition_number = “”

Book class composition

```
class Author:
    def __init__(self):
        self.first_name = ""
        ...

class Edition:
    def __init__(self):
        self.title = ""
        self.authors = [Author(...), ]
        self.isHardcover = True
        ...

class Book:
    def __init__(self):
        self.editions = [Edition(...),]
```

Book class composition

```
books = []
b = Book()
edition1 = Edition(…)
b.editions.append(edition1)
books.append(b)
```

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!

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

Due: February 10, 2015