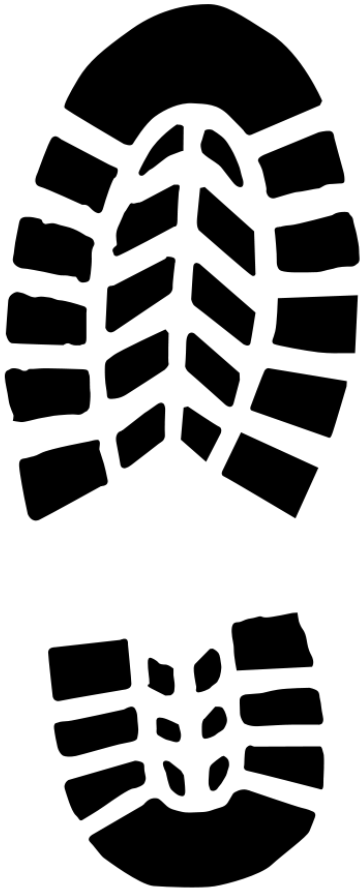


# Welcome to

# Python 2

## Session #5

Michael Purcaro, Chris MacKay,
Nick Hathaway, and the GSBS Bootstrappers

February 2014

michael.purcaro@umassmed.edu

# Building Blocks: modules

- To more easily reuse code, functions and classes can be placed in separate files
- Each file is called a **module**
  - A module is a file containing Python definitions and statements.
  - The file name is the module name with the suffix .py appended.
  - (Within a module, the module's name (as a string) is available as the value of the global variable __name__.)

https://docs.python.org/2/tutorial/modules.html

# Building Blocks: modules

- A module can contain executable statements as well as function definitions.
  - These statements are intended to initialize the module.
  - They are executed only the first time the module name is encountered in an import statement.
  - They are also run if the file is executed as a script.
- To reuse code from the other files, use the **import** command

https://docs.python.org/2/tutorial/modules.html

# Building Blocks: modules

- To reuse code from the other files, use **import**:

dna_sequence.py

```
from collections import defaultdict

class DNASequence(object):
    def __init__(self, sequence, id):
        self.sequence = sequence

    def transcribe(self):
        rna = self.sequence.replace('T','U')
        return rna
```

# Building Blocks: modules

- To use the DNASequence in another file:

hamm.py

```
from dna_sequence import DNASequence

ds = DNASequence(...)
```

# Problem 5: HAMM

- Given two strings s and t of equal length, the Hamming distance between s and t, denoted dH(s,t), is the number of corresponding symbols that differ in s and t.

GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT

The Hamming distance between these two strings is 7. Mismatched symbols are colored red.

- **Given**: Two DNA strings s and t of equal length (not exceeding 1 kbp).
- **Return**: The Hamming distance dH(s,t).
- **Note**: implement the hamm() method inside of DNASequence, but test the method from another file!

# Building Blocks: argparse

- How do we pass “arguments” (i.e. input from the user at the command line) when running the script?

```
python hamm.py --file fileName.txt
```

```
import argparse
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--remote',
                        action="store_true",
                default=False)
    parser.add_argument('--file', type=str)
    parser.add_argument('dirs', type=str,
                            nargs='*')
    return parser.parse_args()
```

# Building Blocks: argparse

```
args = parse_args()
print "arguments: file:", args.file
print "arguments: booleanExample:", args.booleanExample
print "arguments: remainingArguments:", args.remainingArguments
```

```
python hamm.py
```

# Building Blocks: argparse

```
python hamm.py --file fileName.txt
```

```
print "arguments: file:", args.file
```

# Building Blocks: argparse

```
python hamm.py --booleanExample
```

```
print "arguments: booleanExample:",
    args.booleanExample
```

# Building Blocks: argparse

```
python hamm.py a.txt b.txt


print "arguments: remainingArguments:",
     args.remainingArguments
```

# Exercise

- Add argument parsing to hamm.txt
  - Allow the user to specify a filename to read two DNA sequences from via “--file”

# Problem 6: Translating RNA into Protein

- The 20 commonly occurring amino acids are abbreviated by using 20 letters from the English alphabet (all letters except for B, J, O, U, X, and Z).
  - **Protein strings** are constructed from these 20 symbols.
  - Henceforth, the term **genetic string** will incorporate protein strings along with DNA strings and RNA strings.
- The RNA codon table dictates the details regarding the encoding of specific codons into the amino acid alphabet.
- **Given:** An RNA string s corresponding to a strand of mRNA (of length at most 10 kbp).
- **Return:** The protein string encoded by s.
- Help: codon table at
  - http://bioinfo.umassmed.edu/bootstrappers/bootstrappers-courses/python2/lecture4/resources/codon_table.txt

# Problem 7: RNA Splicing

- After identifying the exons and introns of an RNA string, we only need to delete the introns and concatenate the exons to form a new string ready for translation.
- **Given:** A DNA string s (of length at most 1 kbp) and a collection of substrings of s acting as introns. (All strings are given in FASTA format.)
- **Return:** A protein string resulting from transcribing and translating the exons of s.
- **Note**: Only one solution will exist for the dataset provided.

# 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

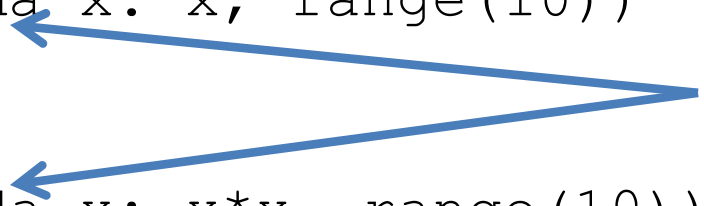
- Another way to build a list
- Applies a function to every element in a list

```
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
```

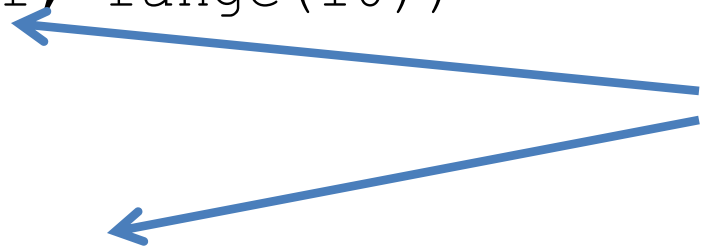
# Building Blocks: map

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

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

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
```



"anonymous" function!
➔ Function with no name



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

# Building Blocks: map

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

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

print len(allPeaks)
```

# Building Blocks: map

```
chrs = map(lambda x: int(x[3:]),
           ["chr1", "chr2", "chr3"])

print chrs
```

# 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)
```

# 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
```