



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__`.)

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

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: HAMMM

- 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 .

G A G C C T A C T A A C G G G A T
C A T C G T A A T G A C G G C C T

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!

Problem 5: HAMMM

```
class DNASequence(object):  
    def __init__(self, sequence):  
        self.sequence = sequence  
  
    def hamm(self, b):  
        if len(self.sequence) != len(b.sequence):  
            raise Exception("strings not equal")  
  
        count = 0  
        for i in xrange(len(self.sequence)):  
            if self.sequence[i] != b.sequence[i]:  
                count += 1  
        return count
```

Problem 5: HAMMM

```
from dna_sequence import DNASequence

def test():
    s1 = "GAGCCTACTAACGGGAT"
    s2 = "CATCGTAATGACGGCCT"
    out = 7
    ds1 = DNASequence(s1)
    ds2 = DNASequence(s2)
    t = ds1.amm(ds2)
    if out == t:
        print "HAMMM: PASSED"
    else:
        print "HAMMM: FAILED!"
```


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
```

Assume Python
version >2.7!

```
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
```

```
print "arguments: file:", args.file  
arguments: file: None
```

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

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

Building Blocks: argparse

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

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

Building Blocks: argparse

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

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

Building Blocks: argparse

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

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

Building Blocks: argparse

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

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

```
arguments: booleanExample: True
```

Building Blocks: argparse

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

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


Building Blocks: argparse

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

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

```
arguments: remainingArguments: ['a.txt', 'b.txt']
```

Exercise

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

Exercise

```
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--file', type=str)
    return parser.parse_args()

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

if not args.file:
    test()
else:
    with open(args.file) as f:
        lines = f.read().split("\n")
        ds1 = DNASequence(lines[0])
        ds2 = DNASequence(lines[1])
        print "Hamming distance:", ds1.hamm(ds2)
```

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 6: Translating RNA into Protein

```
import urllib2

class RNAsequence:
    def __init__(self, sequence):
        self.sequence = sequence

        codonUrl = "http://bioinfo.umassmed.edu/bootstrappers/bootstrappers-
courses/python2/lecture4/resources/codon_table.txt"
        self.codonTable = {}
        for line in urllib2.urlopen(codonUrl).read().split("\n"):
            toks = line.split()
            self.codonTable[toks[0]] = toks[1]

    def translate(self):
        protein = []
        for i in xrange(0, len(self.sequence), 3):
            codon = self.sequence[i : i+3]
            if codon in self.codonTable:
                aa = self.codonTable[codon]
                if "STOP" == aa:
                    break
                protein.append(aa)
        return "".join(protein)
```

```
s = "AUGGCCAUGGCGCC..."
rna = RNAsequence(s)
protein = rna.translate()
```

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.