Students Object to Classes. “They serve no function and we disagree with the methods,” say students.

(Claremont, AP): Students in CS 5 say that they object to classes. “We’re overloaded!” said one student, “and we want to underscore our concerns.” Another student spokesperson said “The professors are def __init__ely hoping this is something that will just float away, but they can’t string us along forever. We have a long list of issues and if the profs don’t understand them, they should look them up in a dictionary,” said a student. “We sure wish the students were mutable!” said one professor. Students and professors eventually agreed on a tuple of ways to __repr__ their relationship.
Human evolution homework revisited...

Orangutan  | Gorilla  | Human  | Common chimpanzee | Bonobo

~15 MYA

~7 MYA

~6 MYA

~3 MYA

~15 MYA
Out of Africa vs. multiregional origin of modern humans

- H. sapiens in Africa
- H. sapiens in Europe
- H. sapiens in Asia

Intermediate species possible

- H. erectus in Africa
- H. erectus in Europe
- H. erectus in Asia

H. sapiens in Africa
H. sapiens in Europe
H. sapiens in Asia

H. erectus in Africa
H. erectus in Europe
H. erectus in Asia
Data consistent with out of Africa
Data consistent with out of Africa
Homework: Markov text generation

1st order

Training text:

I like cookies. I like spam. I am happy. Spam is good.

Learning Phase:

Starters:     [ ("I",), ("I",), ("I",), ("Spam" ,)]
Dictionary:

{ ("I" ,): [ "like" , "like" , "am" ],
  ("like" ,): [ "cookies." , "spam." ],
  ("cookies." ,): [ "I" ],
  ("spam." ,): [ "I" ],
  ("am" ,): [ "happy." ],
  ("happy." ,): [ "Spam" ] ...}
2nd order

Training text:

I like cookies. I like spam. I am happy. Spam is very good.

Learning Phase:

Starters: [(“I”, “like”), ("I", "like"), ("I", "am"), ("Spam", "is")]

Dictionary: {
  ("I", "like"): ["cookies.", "spam."],
  ("I", "am"): ["happy."],
  ("is", "very"): ["good."],
  ...
}
Markov models in biology

- Gene finding
- Sequence similarity search
- Nucleotide substitution models
- Modeling animal behavior
Markov models in bio 52...

Cholera comparative genomics

AAAAAA: 0.048  AAAAAC: 0.021  AAAAAG: 0.013  AAAAAT: 0.019...
AACAAA: 0.029  AACAAC: 0.021  AACAAG: 0.023  AACAAT: 0.031...
AAGAAA: 0.057  AAGAAC: 0.017  AAGAAG: 0.033  AAGAAT: 0.020...
AATAAA: 0.049  AATAAC: 0.016  AATAAG: 0.016  AATAAT: 0.034...
ACAAAA: 0.022  ACAAAC: 0.015  ACAAAG: 0.011  ACAAAT: 0.033...
...

Probabilistic gene finder using a 1st order model on codons
Oops (object-oriented programs) example 1: simulating a population of RNA organisms

An RNA ‘organism’ AGAAAAAACAA

Fitness (probability of reproducing) depends on number of secondary structure pairing interactions.
Selection and reproduction over a series of generations

- 1/3 of sequences with most pairing interactions selected to form “breeding population”.

- Sample with replacement to obtain parent sequences
- Replicate these with mutation for form the next generation
def sim(seqLen, popSize, numGens):
    '''Evolve RNA stings over numGens generations.'''

    # get initial population
    popL = initialPop(popSize, seqLen)
    print('Initial population fitness', meanFitness(popL))

    # evolve...
    for i in range(numGens):
        popL = nextGen(popL)

    # print mean fitness of final population
    print('Final population fitness', meanFitness(popL))
    return popL
import random
def nextGen(popL):
    '''Given a population, find the most fit 1/3 and use these to reproduce the next generation.'''

    popL.sort(reverse=True) # sort high to low

    breedL = popL[:int(len(popL)/3)]

    newPopL=[]
    for i in range(len(popL)):
        parent = random.choice(breedL)
        newPopL.append(parent.replicate())
    return newPopL

Getting the next generation

For this to work, rnaOrg class must have __eq__ and __lt__
class rnaOrg:
    def __init__(self, seq):
        self.seq = seq

    def __repr__(self):
        ...

    def __eq__(self, other):
        ...

    def __lt__(self, other):
        # less than

You can assume the replicate method has already been written (you don’t need to write it).
class rnaOrg:
    def __init__(self, seq):
        self.seq = seq
        self.fitness = self.getFitness()

    def getFitness(self):
        '''Return the total number of pairing interactions in seq.'''
        return mfold5(self.seq,{})

    def __repr__(self):
        return str(self.fitness) + " " + self.seq

    def __eq__(self, other):
        return self.fitness == other.fitness

    def __lt__(self, other):
        return self.fitness < other.fitness
Oops example 2: dates

```python
>>> today = Date(11, 15, 2018)
>>> due = Date(11, 20, 2018)
>>> due - today
5
```
Oops example 2: dates

```python
def __sub__(self, other):
    blah, blah, blah

class Date:
    def __init__(self, day, month, year):
        ...  
    def __sub__(self, other):
        blah, blah, blah

>>> today = Date(11, 15, 2018)
>>> due = Date(11, 20, 2018)
>>> due - today
5
```
Oops example 2: dates

```python
>>> today = Date(11, 15, 2018)
>>> due = Date(11, 20, 2018)
>>> if due > today:
    print(“let’s watch a movie!”)
```
class Date:
    def __init__(self, m, d, y):
        self.month = m
        self.day = d
        self.year = y

    def __sub__(self, other):
    def __gt__(self, other):

>>> d = Date(11, 15, 2018)
Another implementation...

class Date:
    def __init__(self, m, d, y):
        self.daysSinceJanFirst1900 = some_funky_math_here!

    def __sub__(self, other):
    def __gt__(self, other):

>>> d = Date(11, 15, 2018)

Why would any sane person want to store the date as the number of days since January 1, 1900?
Converting in and out of an internal representation

class Date:
    def __init__(self, m, d, y):
        self.daysSinceJanFirst1900 = self.getDaysSince1900(m, d, y)

    def getDaysSince1900(self, m, d, y):
        funky math here

    def getMonthDayYear(self):
        funky math in reverse here

>>> d = Date(11, 15, 2018)
>>> d.getMonthDayYear ()
(11, 15, 2018)
Date “Abstraction”

Date
    __init__(self, month, day, year)
getDaysSince1900(self, m, d, y)
getMonthDayYear(self)
==, >, <, >=, <=, +, -
The Age of Abstraction

A final oops example: protein protein interaction networks
Some input data

edgeL = [
    (gene4634, gene2542),
    (gene2351, gene3807),
    (gene207, gene2331),
    (gene2180, gene4867),
    (gene4224, gene2073),
    (gene4128, gene1902),
    (gene785, gene4093),
    (gene3879, gene1734),
    (gene4906, gene2255),
]
def isConnected(gene1, gene2, edgeL):
    '''Are gene1 and gene2 connected in edgeL? Returns boolean.'''

def isConnected(gene1, gene2, edgeL):
    '''Are gene1 and gene2 connected in edgeL? Returns boolean.'''

    for edgeGeneA, edgeGeneB in edgeL:
        if edgeGeneA == gene1 and edgeGeneB == gene2:
            return True
        elif edgeGeneA == gene2 and edgeGeneB == gene1:
            return True

    return False
What if the network is really big and we have a lot of queries?

def exampleEdgeCheck(queryEdgeL, edgeL):
    '''Look in for queryEdgeL in edgeL. Return a list of those present.'''

    presentL = []
    for q1, q2 in queryEdgeL:
        if isConnected(q1, q2, edgeL):
            presentL.append((q1, q2))
    return presentL

The technical term is hairball.

A cancer related protein-protein interaction network

A network class

```python
class Network:
    
def __init__(self, edgeL):
        self.nodeConnectD = {}
        for geneA, geneB in edgeL:
            self.addEdge(geneA, geneB)

    def addEdge(self, geneA, geneB):
        '''Add edge to network.'''
        if geneA in self.nodeConnectD:
            self.nodeConnectD[geneA].append(geneB)
        else:
            self.nodeConnectD[geneA] = [geneB]

        if geneB in self.nodeConnectD:
            self.nodeConnectD[geneB].append(geneA)
        else:
            self.nodeConnectD[geneB] = [geneA]
```

nodeConnectD is an attribute that stores the network:
- Keys are genes (proteins)
- Values are list of other genes a given gene is connected to
Write an isConnected method for this Network class.

def isConnected(self, gene1, gene2):
    '''Are gene1 and gene2 connected? Returns boolean.'''
Write an isConnected method for this Network class.

def isConnected(self, gene1, gene2):
    '''Are gene1 and gene2 connected? Returns boolean.'''
    if gene1 in self.nodeConnectD:
        if gene2 in self.nodeConnectD[gene1]:
            return True
    return False
Try the network version out...

```python
def exampleEdgeCheckNetwork(queryEdgeL, net):
    '''Look in for queryEdgeL in edgeL. Return a list of those present.'''
    presentL = []
    for q1, q2 in queryEdgeL:
        if net.isConnected(q1, q2):
            presentL.append((q1, q2))
    return presentL

>>> net = Network(edgeL)
>>> exampleEdgeCheckNetwork(queryEdgeL, net) # Demo
```
See you in lab