CS 5 Black Today

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. “We want to underscore our concerns,” said another. “This may seem like a local issue, but actually it has global scope.” The professors for the course said that there is an array of variables at play but refused to list them all. “The professors are def __init__ely hoping this as something that will just float away, but they can’t string us along forever. If they don’t understand our references, they should look them up in the 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.

News Briefs

P.I.T. mathematician proves that $1+1 = 2.01$ Implications unclear. (p. 42)

Foundations of CS proved to be wrong by P.I.T scientists. All software immediately stops working. (p. 42)

Today’s lecture sponsored by ants. “We’re NOT antagonistic to objects and classes,” says ant rep (p. 42)

Weather: Cold and warm days ahead over next 12 months
Sports: Python programming newest winter Olympic sport
ROOTER: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

ABSTRACT

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. INTRODUCTION

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-hailed autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in \( \Omega((n + \log n)) \) time [22]. In the end, we conclude.

II. ARCHITECTURE
Video!
**k^{th} Order Markov Processes**

**Training File:** “I like spam. I like toast and spam. I eat ben and jerry’s ice cream too.”

**First order Markov Dictionary:**
- I : like, like, eat
- like : spam, toast
- spam. : I, I
- and : spam, jerry’s
- MORE ENTRIES...

**Generating “random” text:**
- “I like spam. I like spam.”
- “I eat ben and spam. I like toast and jerry’s ice cream too.”

Andrey Markov
1856-1922
Training File: Wikipedia essay on Huffman Compression

First order Markov sentences generated...

“Huffman was a source symbol.”

“Huffman became a known as a character in a particular symbol frequencies agree with those used for each possible value of Engineering.”
kth Order Markov Processes

Training File: “I like spam. I like toast and spam. I eat ben and jerry’s ice cream too.”

First order Markov Dictionary:
- I : like, like, eat
- like : spam, toast
- spam. : I, I
- and : spam, jerry’s

Second order Markov Dictionary:
- I like : spam., toast
- like spam. : I
- spam. I : like, eat

Andrey Markov
1856-1922
Training File: Wikipedia essay on Huffman Compression

Second order Markov sentences generated...

"Huffman coding is such a code is not produced by Huffman's algorithm."

"Huffman was able to design the most common characters using shorter strings of bits than are used for lossless data compression."
OOPs! (Object-Oriented Programs)

```python
>>> assigned = Date(1, 27, 2010)
>>> due = Date(2, 1, 2010)
>>> due - assigned
5

>> if due > assigned:
    print "Go watch a movie!"
```
One implementation

class Date:
    def __init__(self, m, d, y):
        self.month = m
        self.day = d
        self.year = y

>>> d = Date(1, 21, 1969)
Another implementation...

class Date:
    def __init__(self, m, d, y):
        self.daysSince1900 = ...

>>> d = Date(1, 21, 1969)

Why would any sane person want to store the date as the number of days since January 1, 1900?
Getters and Setters

class Date:
    def __init__(self, m, d, y):
        self._daysSince1900 = ...

    def setDay(self, d):
        if d <= 0 or d > 31:
            ...
        else:
            self._dayCount = ...

>>> d = Date(1, 21, 1969)
>>> d.setDay(28)    # SETTER
>>> x = d.getDay()  # GETTER
Date "Abstraction"

Date
    __init__(self, month, day, year)
    setDay(self, day)
    setMonth(self, month)
    setYear(self, year)
    getDay(self)
    getMonth(self)
    getYear(self)
    ==, >, <, >=, <=, +, -
The Advantage of Abstraction

import turtle
import math
import Date

turtle.forward(100)
print math.cos(math.pi)
today = Date.Date(11, 9, 2011)
Another Point...

class Point:
    def __init__(self, InputX, InputY):
        self.x = InputX
        self.y = InputY

def __repr__(self):
    return "(" + str(self.x) + "," + str(self.y) + ")"

def __eq__(self, other):
    return self.x == other.x and self.y == other.y

>>> P1 = Point(1.0, 2.0)
>>> P2 = Point(1.0, 2.0)
>>> P1
???
>>> P1 == P2
???
class Point:
    def __init__(self, InputX, InputY):
        self.x = InputX
        self.y = InputY

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

class Line:
    def __init__(self, Point1, Point2):
        self.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)

    def __repr__(self):
        return "(" + str(self.Point1.x) + "," + str(self.Point1.y) + ")" + " slope: " + str(self.slope) + " y-intercept: " + str(self.yintercept)

    def __eq__(self, other):
        return self.Point1 == other.Point1 and self.Point2 == other.Point2
Thinking Linearly

class Point:
    def __init__(self, InputX, InputY):
        self.x = InputX
        self.y = InputY

    def __repr__(self):
        return "({}, {})".format(self.x, self.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

class Line:
    def __init__(self, Point1, Point2):
        self.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)

    def __repr__(self):
        return "y = {} \times + \{\}".format(self.slope, self.yintercept)

    def __eq__(self, other):
        return self.slope == other.slope and self.yintercept == other.yintercept

>>> P1 = Point(1.0, 2.0)
>>> P2 = Point(2.0, 3.0)
>>> L1 = Line(P1, P2)
>>> L1
y = 1.0 x + 1.0
>>> P3 = Point(3.0, 4.0)
>>> P4 = Point(42.0, 43.0)
>>> L2 = Line(P3, P4)
>>> L1 == L2
True
>>> from Point import *
>>> p1 = Point(0, 1)
>>> p2 = Point(1, 2)
>>> L1 = Line(p1, p2)
>>> p3 = Point(2, 0)
>>> p4 = Point(0, 2)
>>> L2 = Line(p3, p4)
>>> L1.parallel(L2)
False
>>> L1.intersection(L2)
(0.5,1.5)

class Line:
    def __init__ (self, Point1, Point2):
        self.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)

    def __repr__(self):
        return "y = " + str(self.slope) + " x + " + str(self.yintercept)

    def __eq__(self, other):
        return self.slope == other.slope and self.yintercept == other.yintercept

    def parallel(self, other):
        return False

    def intersection(self, other):
        if Line.parallel(self, other):
            return None
        else:
            x = (self.yintercept - other.yintercept)/(other.slope - self.slope)
            y = self.slope * x + self.yintercept

Can you think of another way of writing this line?
>>> from Point import *
>>> p1 = Point(0, 1)
>>> p2 = Point(1, 2)
>>> L1 = Line(p1, p2)
>>> p3 = Point(2, 0)
>>> p4 = Point(0, 2)
>>> L2 = Line(p3, p4)
>>> L1.parallel(L2)
False
>>> L1.intersection(L2)
(0.5, 1.5)

class Point:
    def __init__(self, InputX, InputY):
        self.x = 1.0*InputX
        self.y = 1.0*InputY

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

class Line:
    def __init__(self, Point1, Point2):
        self.Point1 = Point1
        self.Point2 = Point2
        self.slope = (Point1.y - Point2.y) / (Point1.x - Point2.x)
        self.yintercept = Point1.y - Point1.x*(Point2.y - Point1.y)/(Point2.x - Point1.x)

    def __repr__(self):
        return "y = " + str(self.slope) + "x + " + str(self.yintercept)

    def __eq__(self, other):
        return self.slope == other.slope and self.yintercept == other.yintercept

    def parallel(self, other):
        return self.slope == other.slope

    def intersection(self, other):
        if Line.parallel(self, other): return None
        else:
            x = (self.yintercept - other.yintercept)/(other.slope - self.slope)
            y = self.slope * x + self.yintercept
            return Point(x, y)
Default Arguments

```python
class Student:
    def __init__(self, firstName, lastName, school="HMC", major = "undeclared")

>>> nick = Student("Nick", "Carter")
>>> joe = Student(firstName = "Joe", "Shmo", "PIT")
>>> anna = Student("Anna", "Litik", major="Physics")
>>> elmo = Student("Elmo")
>>> bigBird = Student("Big", "Bird", firstName = "Tweety")
>>> bart = Student(school="PIT", "Bart", "Simpson")
```

In my experience, arguments are usually default of deperson who started them!
Inheritance (inher*ts!)

class Person:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7  # MILITARY TIME

    def __repr__(self):
        return self.firstName + " " + self.lastName

>>> ran = Person("Ran", "Libeskind-Hadas")
>>> ran
Ran Libeskind-Hadas
>>> ran.asleep(2)
True
class Person:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7

    def __repr__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        Person.__init__(self, first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11
    def __repr__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"

>>> s = Student("Sue", "Persmart", 18)
>>> s
Sue Persmart, 18 years old
>>> s.asleep(2)
False
class Person:
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return 0 <= time <= 7

    def __repr__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        Person.__init__(self, first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11

    def __repr__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"

class Mudder(Student):
    def __init__(self, first, last, age, dorm):
        Student.__init__(self, first, last, age)
        self.dorm = dorm

    def asleep(self, time):
        return False

Get some sleep!!!

>>> wally = Mudder(“wally”, “wart”, 42, “west”)
>>> wally
?
>>> wally.asleep()
?
The Dangers of Inheritance
Python forbids personality transplants!
Millisoft “Shapes”

```python
>>> r = Rectangle(100, 50, center=Vector(80, 60), color="blue")
>>> c = Circle(radius=30, color = "red") # default center (0,0)
>>> r.rotate(15) # 15 degree ccw rotation
>>> r.render()
>>> c.render()
```

That rectangle an’t parallel to the x-axis!

Demo ShapesDemo.py
Transformation Matrices

Rotation
Scaling
Translation ???

Rotation matrices make my head spin.
A Matrix Class

>>> m1 = Matrix(0, -1, 1, 0)
>>> m2 = Matrix(1, 2, 3, 4)
>>> m1
0 -1
1 0
>>> m2
1 2
3 4
>>> m1+m2
1 1
4 4
>>> m1*m2
-3 -4
1 2
>>> m1.get(0, 1)
-1
>>> m1.set(1, 0, 42)
I thought that Linear Algebra was the Matrix Class!
Matrix Class

def Matrix:
    """2x2 matrix class""
    def __init__(self, a11=0, a12=0, a21=0, a22=0):
        self.array = [[a11, a12], [a21, a22]]
    
    def __repr__(self)
    
    def set(self, row, column, value):
    
    def get(self, row, column):
    
    def __mul__(self, other):
        """other may be a matrix OR a vector and returns the product of self and other.""
        Blah, blah, blah
        if other.__class__.__name__ == "Matrix":
            blah, blah, blah
        else:  # it’s HOPEFULLY a Vector!

class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    
    def magnitude(self):
        blah, blah, blah
        return ...
    
    def normalize(self):
        mag = self.magnitude()
        self.x = self.x/mag
        self.y = self.y/mag

I was trying to anticipate how this would be done!
def __mul__(self, other):
    """ if other is a Matrix, returns a Matrix.
    If other is a Vector, returns a Vector.""
    if other.__class__.__name__ == "Matrix":
        result = Matrix()
        for row in range(0, 2):
            for col in range(0, 2):
                entry = 0
                for i in range(0, 2):
                    entry += ______________________
                result.set(row, col, entry)
        return result
    elif other.__class__.__name__ == "Vector":
        x = ______________________
        y = ______________________
        return ______________________
    else:
        print "Can't multiply a matrix by a ", other.__class__.__name__

Fill this in...
def __mul__(self, other):
    """ if other is a Matrix, returns a Matrix. If other is a Vector, returns a Vector."""
    if other.__class__.__name__ == "Matrix":
        result = Matrix()
        for row in range(0, 2):
            for col in range(0, 2):
                # Compute result matrix in the given row and col
                entry = 0
                for i in range(0, 2):
                    entry += self.get(row, i) * other.get(i, col)
                result.set(row, col, entry)
        return result
    elif other.__class__.__name__ == "Vector":
        x = self.get(0, 0) * other.x + self.get(0, 1) * other.y
        y = self.get(1, 0) * other.x + self.get(1, 1) * other.y
        return Vector(x, y)
    else:
        print "Can't multiply a matrix by a ", other.__class__.__name__
import math  # Now we have math.cos(angle), math.sin(angle), etc. Angles are in radians
import turtle
from Matrix import *
from Vector import *

class Shape:
    def __init__(self):
        self.points = []  # List of Vectors!

    def render(self):
        turtle.penup()
        turtle.setposition(self.points[0].x, self.points[0].y)
        turtle.pendown()
        turtle.fillcolor(self.color)
        turtle.pencolor(self.color)
        turtle.begin_fill()
        for vector in self.points[1:]:
            turtle.setposition(vector.x, vector.y)
            turtle.setposition(self.points[0].x, self.points[0].y)
        turtle.end_fill()

    def erase(self):
        temp = self.color
        self.color = "white"
        self.render()
        self.color = temp

    def rotate(self, theta):
        """ Rotate shape by theta degrees """
        theta = math.radians(theta)  # Python thinks in radians

class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):

class Square...  (constructor takes width, optional center, optional color)
class Shape:
    def __init__(self):
        self.points = []
    def render(self):
        def rotate(self, theta):
            """ Rotate shape by theta degrees """
            theta = math.radians(theta) # Python thinks in radians

class Rectangle(Shape):
    def __init__(self, width, height, center=Vector(0, 0), color = "black"):
        SW = Vector(center.x - width/2.0, center.y - height/2.0)
        NW = Vector(center.x - width/2.0, center.y + height/2.0)
        NE = Vector(center.x + width/2.0, center.y + height/2.0)
        SE = Vector(center.x + width/2.0, center.y - height/2.0)
        self.points = [SW, NW, NE, SE]
        self.color = color

class Square
    def __init__(self, width, center=Vector(0, 0), color = "black"):
class Shape:
    
def __init__(self):
        self.points = []
    
def render(self):
        
def rotate(self, theta):
            """ Rotate shape by theta degrees """
            theta = math.radians(theta)
            Do this one last


class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):
        SW = Vector(center.x - width/2.0, center.y - height/2.0)
        NW = Vector(center.x - width/2.0, center.y + height/2.0)
        NE = Vector(center.x + width/2.0, center.y + height/2.0)
        SE = Vector(center.x + width/2.0, center.y - height/2.0)
        self.points = [SW, NW, NE, SE]
        self.color = color


class Square(Rectangle):
    def __init__(self, width, center=Vector(0, 0), color = "black"):
        Rectangle.__init__(self, width, width, center, color)
class Shape:
    def __init__(self):
        self.points = []

    def render(self):
        def rotate(self, theta):
            """Rotate shape by theta degrees ""
            theta = math.radians(theta)  # Python thinks in radians!
            RotationMatrix = Matrix(math.cos(theta), -1*math.sin(theta), math.sin(theta), math.cos(theta))
            NewPoints = []
            for vector in self.points:
                newvector = RotationMatrix * vector
                NewPoints.append(newvector)
            self.points = NewPoints

class Rectangle(Shape):
    def __init__(self, width, height, center = Vector(0, 0), color = "black"):
        SW = Vector(center.x - width/2.0, center.y - height/2.0)
        NW = Vector(center.x - width/2.0, center.y + height/2.0)
        NE = Vector(center.x + width/2.0, center.y + height/2.0)
        SE = Vector(center.x + width/2.0, center.y - height/2.0)
        self.points = [SW, NW, NE, SE]
        self.color = color

class Square(Rectangle):
    def __init__(self, width, center=Vector(0, 0), color = "black"):
        Rectangle.__init__(self, width, width, center, color)
class Shape:
    
    def __init__(self):
        self.points = []

    def render(self):

    def rotate(self, theta):

class Circle (Shape):
    
    def __init__(self, center=Vector(0,0), radius=10, color="black")

    turtle.circle(50)

Inherit render and rotate from Shape?
More “Draw” Tricks

Rotation about an arbitrary point

Homogenous coordinates and translation!

August Mobius