6.00 Handout, Lecture 12
(Not intended to make sense outside of lecture)

```
class CourseList(object):
    def __init__(self, number):
        self.number = number
        self.students = []
    def addStudent(self, who):
        if not who.isStudent():
            raise TypeError('Not a student')
        if who in self.students:
            raise ValueError('Duplicate student')
        self.students.append(who)
    def remStudent(self, who):
        try:
            self.students.remove(who)
        except:
            print str(who) + ' not in ' + self.number
    def allStudents(self):
        for s in self.students:
            yield s
    def ugs(self):
        indx = 0
        while indx < len(self.students):
            if type(self.students[indx]) == UG:
                yield self.students[indx]
            indx += 1
```

```
m1 = MITPerson('Barbara Beaver')
ug1 = UG('Jane Doe')
ug2 = UG('John Doe')
g1 = G('Mitch Peabody')   t
g2 = G('Ryan Jackson')
g3 = G('Sarina Canelake')
SixHundred = CourseList('6.00')
SixHundred.addStudent(ug1)
SixHundred.addStudent(g1)
SixHundred.addStudent(ug2)
try:
    SixHundred.addStudent(m1)
except:
    print 'Whoops'
print SixHundred #Perhaps not
what one expected
SixHundred.remStudent(g3)
print 'Students'
for s in
SixHundred.allStudents():
    print s
print 'Students Squared'
for s in
SixHundred.allStudents():
    for s1 in
SixHundred.allStudents():
        print s, s1
print 'Undergraduates'
for u in SixHundred.ugs():
    print u
```

```
import random

class Location(object):
    def __init__(self, x, y):
        """x and y are floats"""
        self.x = x
        self.y = y
    def move(self, deltaX, deltaY):
        """deltaX and deltaY are floats"""
        return Location(self.x + deltaX, self.y + deltaY)
    def getX(self):
        return self.x
    def getY(self):
        return self.y
    def distFrom(self, other):
        ox = other.x
        oy = other.y
        xDist = self.x - ox
        yDist = self.y - oy
        return (xDist**2 + yDist**2)**0.5
    def __str__(self):
        return '<' + str(self.x) + ', ' + str(self.y) + '>'
```

```
class Field(object):
    def __init__(self):
        self.drunks = {}
    def addDrunk(self, drunk, loc):
        if drunk in self.drunks:
            raise ValueError('Duplicate drunk')
        else:
            self.drunks[drunk] = loc
    def moveDrunk(self, drunk):
        if not drunk in self.drunks:
            raise ValueError('Drunk not in field')
        xDist, yDist = drunk.takeStep()
        self.drunks[drunk] = self.drunks[drunk].move(xDist, yDist)
    def getLoc(self, drunk):
        if not drunk in self.drunks:
            raise ValueError('Drunk not in field')
        return self.drunks[drunk]

class Drunk(object):
    def __init__(self, name):
        self.name = name
    def takeStep(self):
        stepChoices = [(0,1), (0,-1), (1, 0), (-1, 0)]
        return random.choice(stepChoices)
    def __str__(self):
        return 'This drunk is named ' + self.name

def walk(f, d, numSteps):
    start = f.getLoc(d)
    for s in range(numSteps):
        f.moveDrunk(d)
    return(start.distFrom(f.getLoc(d)))

def simWalks(numSteps, numTrials):
    homer = Drunk('Homer')
    origin = Location(0, 0)
    distances = []
    for t in range(numTrials):
        f = Field()
        f.addDrunk(homer, origin)
        distances.append(walk(f, homer, numTrials))
    return distances

def drunkTest(numTrials):
    for numSteps in [10, 100, 1000, 10000, 100000]:
        distances = simWalks(numSteps, numTrials)
        print 'Random walk of ' + str(numSteps) + ' steps'
        print '  Mean =', sum(distances)/len(distances)
        print '  Max =', max(distances), 'Min =', min(distances)
```

6.00SC Introduction to Computer Science and Programming
Spring 2011