Random Walks and More Plotting

How Far Will a Drunk Get?

▪Simulated, by hand, a walk in last lecture

▪Process too labor intensive to be practical for more than a few steps

▪But we can write a program to simulate lots of steps

Structure of Simulation

- Simulate one walks of k steps

- Simulate n such walks

- Report average distance from origin

First, Some Useful Abstractions

- Location—a place

- Field—a collection of places and drunks

- Drunk—somebody who wanders from place to place in a field

Class Location, part 1

```
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
```

Class Location, continued

```
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, part 1

```
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 getLoc(self, drunk):
        if drunk not in self.drunks:
            raise ValueError('Drunk not in field')
        return self.drunks[drunk]
```

Class Field, continued

```
def moveDrunk(self, drunk):
    if drunk not in self.drunks:
        raise ValueError('Drunk not in field')
    xDist, yDist = drunk.takeStep()
    currentLocation = self.drunks[drunk]
    #use move method of Location to get new location
    self.drunks[drunk] =\
        currentLocation.move(xDist, yDist)
```

Notable Aspects of Class Field

- A mapping of drunks to locations

- Unbounded size

- Allows multiple drunks
  - With no constraints about how they relate to each other

Class Drunk

```
class Drunk(object):
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return 'This drunk is named ' + self.name
```

Not intended to be useful on its own

A base class to be inherited

Two Subclasses of Drunk

- The "usual" drunk, who wanders around at random

- The "I hate winter" drunk, who tries to move southward

Two Kinds of Drunks

```python
import random

class UsualDrunk(Drunk):
    def takeStep(self):
        stepChoices =\
            [(0.0,1.0),(0.0,-1.0),(1.0,0.0),(-1.0,0.0)]
        return random.choice(stepChoices)

class ColdDrunk(Drunk):
    def takeStep(self):
        stepChoices =\
            [(0.0,0.9),(0.0,-1.1),(1.0,0.0),(-1.0,0.0)]
        return random.choice(stepChoices)
```