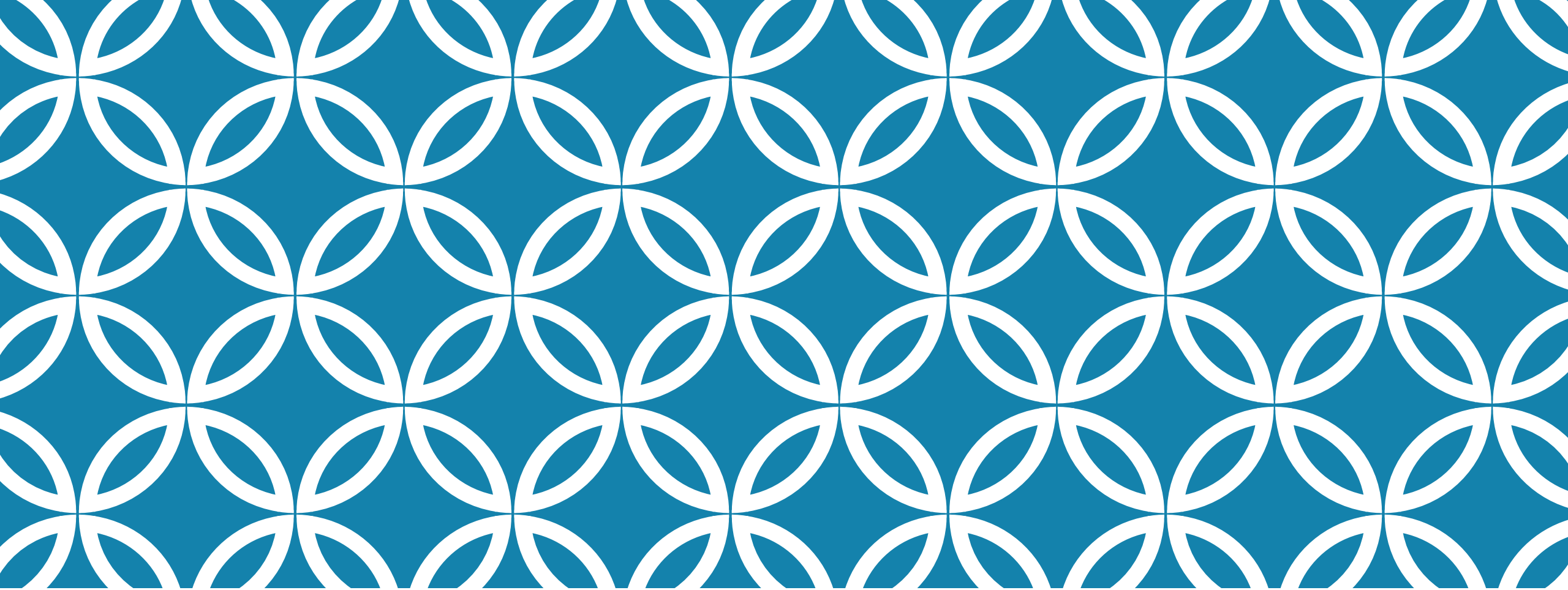


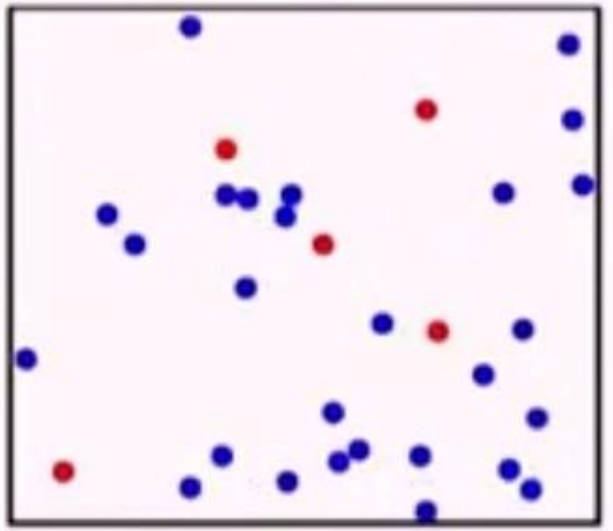
DATA SCIENCE RANDOM WALK LECTURFE

Win+w

Lecture 5: Random Walks

Why Random Walks?

- Random walks are important in many domains
  - Understanding the stock market (maybe)
  - Modeling diffusion processes
  - Etc.
- Good illustration of how to use simulations to understand things
- Excuse to cover some important programming topics
  - Practice with classes
  - Practice with plotting



Class Location, part 1

Immutable type

```
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 Drunk

```
class Drunk(object):
    def __init__(self, name = None):
        """Assumes name is a str"""
        self.name = name

    def __str__(self):
        if self != None:
            return self.name
        return 'Anonymous'
```

Two Subclasses of Drunk

- The "usual" drunk, who wanders around at random
- The "masochistic" drunk, who tries to move northward

Two Kinds of Drunks

```
import random

class UsualDrunk(Drunk):
    def takeStep(self):
        stepChoices = [(0,1), (0,-1), (1, 0), (-1, 0)]
        return random.choice(stepChoices)

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

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()
    #use move method of Location to get new location
    self.drunks[drunk] =\
        self.drunks[drunk].move(xDist, yDist)
```

Simulating a Single Walk

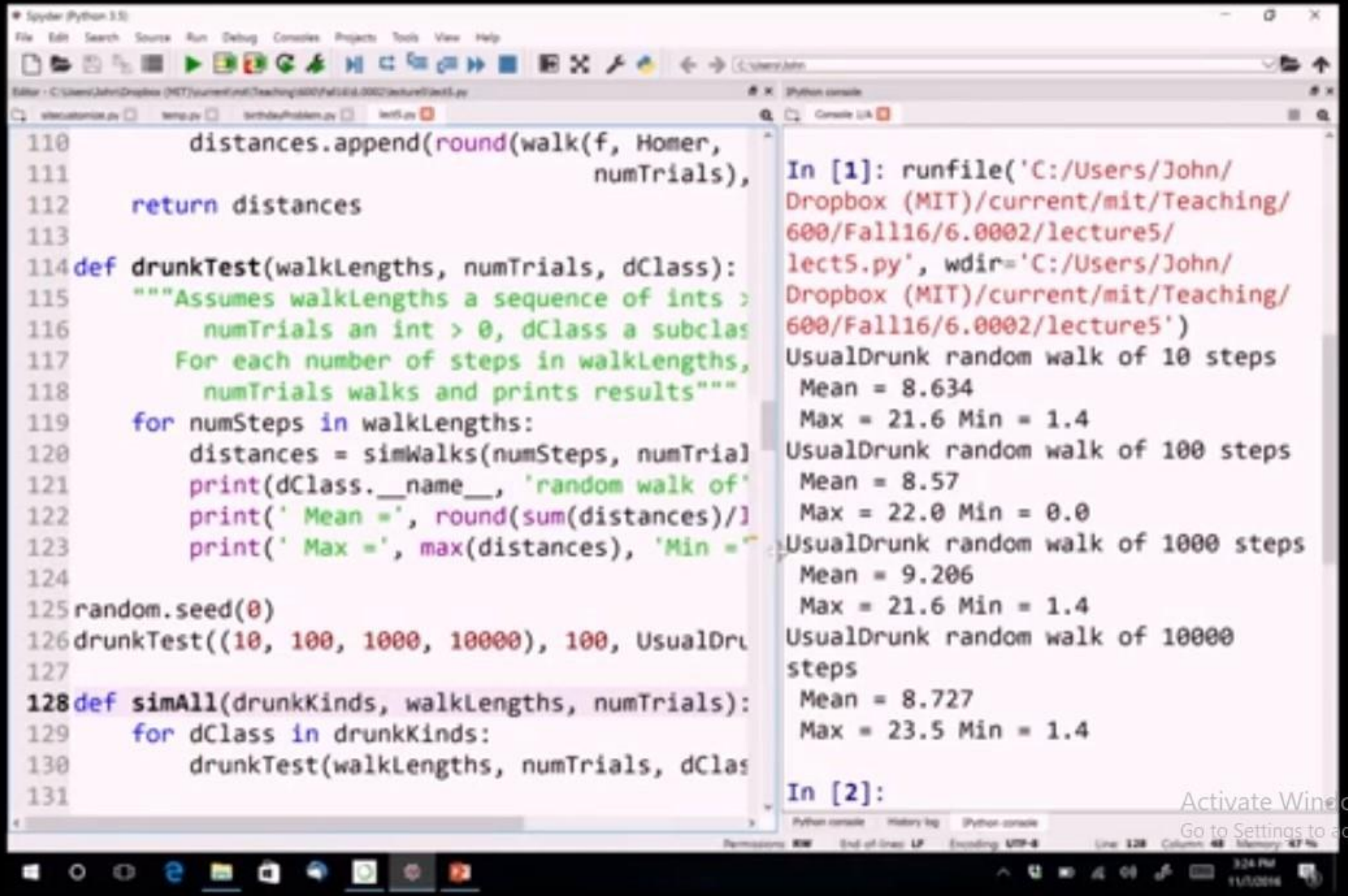
```
def walk(f, d, numSteps):
    """Assumes: f a Field, d a Drunk in f, and
       numSteps an int >= 0.
       Moves d numSteps times; returns the distance
       between the final location and the location
       at the start of the  walk."""
    start = f.getLoc(d)
    for s in range(numSteps):
        f.moveDrunk(d)
    return start.distFrom(f.getLoc(d))
```

Simulating Multiple Walks

```
def simWalks(numSteps, numTrials, dClass):
    """Assumes numSteps an int >= 0, numTrials an
         int > 0, dClass a subclass of Drunk
       Simulates numTrials walks of numSteps steps
         each. Returns a list of the final distances
         for each trial"""
    Homer = dClass()
    origin = Location(0, 0)
    distances = []
    for t in range(numTrials):
        f = Field()
        f.addDrunk(Homer, origin)
        distances.append(round(walk(f, Homer,
                                    numTrials), 1))
    return distances
```

Putting It All Together

```
def drunkTest(walkLengths, numTrials, dClass):
    """Assumes walkLengths a sequence of ints >= 0
         numTrials an int > 0,
         dClass a subclass of Drunk
       For each number of steps in walkLengths,
         runs simWalks with numTrials walks and
         prints results"""
    for numSteps in walkLengths:
        distances = simWalks(numSteps, numTrials,
                             dClass)
        print(dClass.__name__, 'random walk of',
              numSteps, 'steps')
        print(' Mean =',
              round(sum(distances)/len(distances), 4))
        print(' Max =', max(distances),
              'Min =', min(distances))
```


distances.append(round(walk(f, Homer,
numTrials),
return distances
def drunkTest(walkLengths, numTrials, dClass):
"""Assumes walkLengths a sequence of ints >
numTrials an int > 0, dClass a subclas
For each number of steps in walkLengths,
numTrials walks and prints results"""
for numSteps in walkLengths:
distances = simWalks(numSteps, numTrial
print(dClass.__name__, 'random walk of'
print(' Mean =', round(sum(distances)/1
print(' Max =', max(distances), 'Min ='
random.seed(0)
drunkTest((10, 100, 1000, 10000), 100, UsualDru
def simAll(drunkKinds, walkLengths, numTrials):
for dClass in drunkKinds:
drunkTest(walkLengths, numTrials, dClas
In [1]: runfile('C:/Users/John/
Dropbox (MIT)/current/mit/Teaching/
600/Fall16/6.0002/lecture5/
lect5.py', wdir='C:/Users/John/
Dropbox (MIT)/current/mit/Teaching/
600/Fall16/6.0002/lecture5')
UsualDrunk random walk of 10 steps
Mean = 8.634
Max = 21.6 Min = 1.4
UsualDrunk random walk of 100 steps
Mean = 8.57
Max = 22.0 Min = 0.0
UsualDrunk random walk of 1000 steps
Mean = 9.206
Max = 21.6 Min = 1.4
UsualDrunk random walk of 10000
steps
Mean = 8.727
Max = 23.5 Min = 1.4
In [2]:
Activate Windows
Go to Settings to activate Windows.

Let's Try It

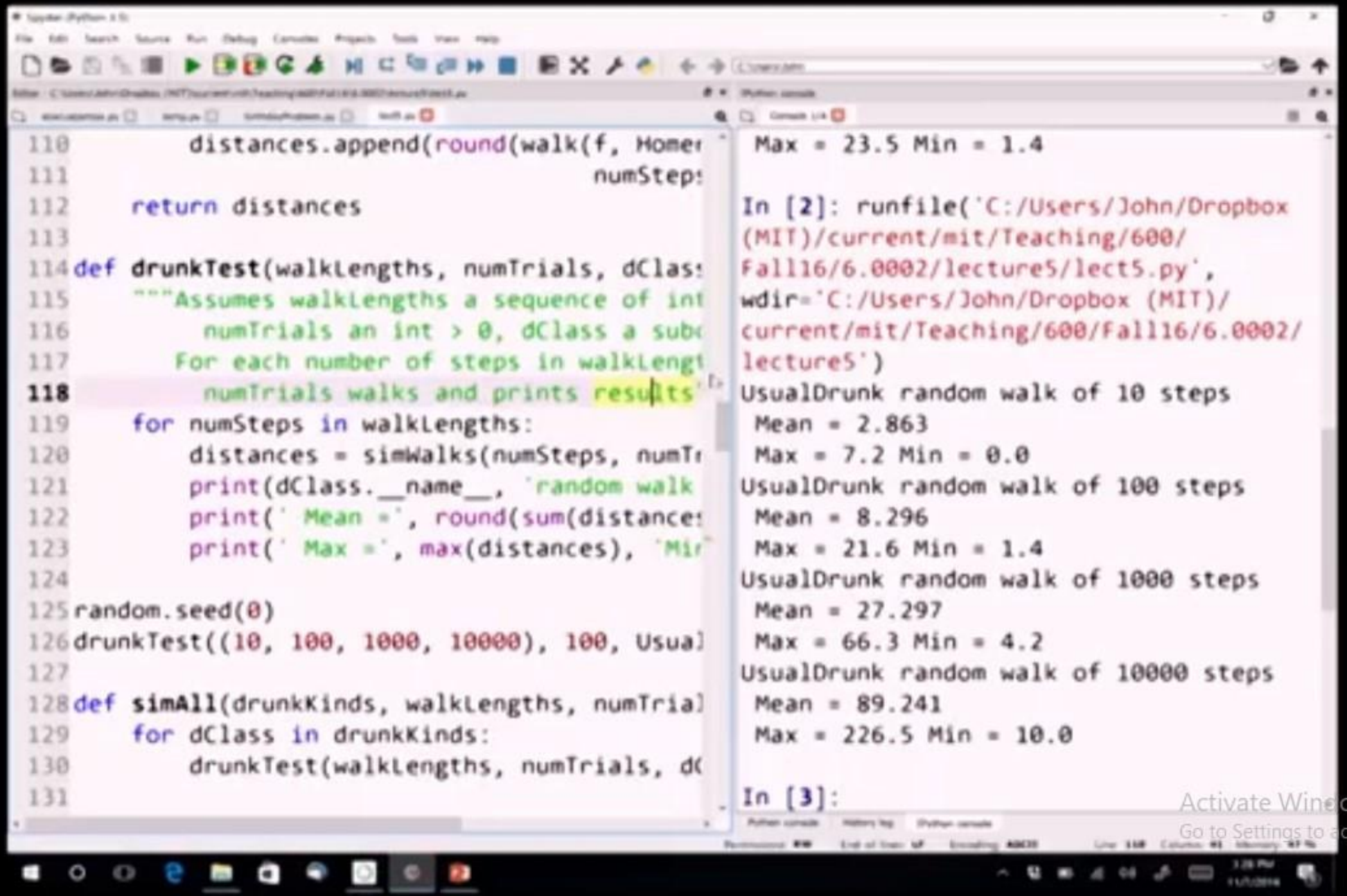
```
drunkTest((10, 100, 1000, 10000), 100,
          UsualDrunk)

UsualDrunk random walk of 10 steps
 Mean = 8.634
 Max = 21.6 Min = 1.4
UsualDrunk random walk of 100 steps
 Mean = 8.57
 Max = 22.0 Min = 0.0
UsualDrunk random walk of 1000 steps
 Mean = 9.206
 Max = 21.6 Min = 1.4
UsualDrunk random walk of 10000 steps
 Mean = 8.727
 Max = 23.5 Min = 1.4
```

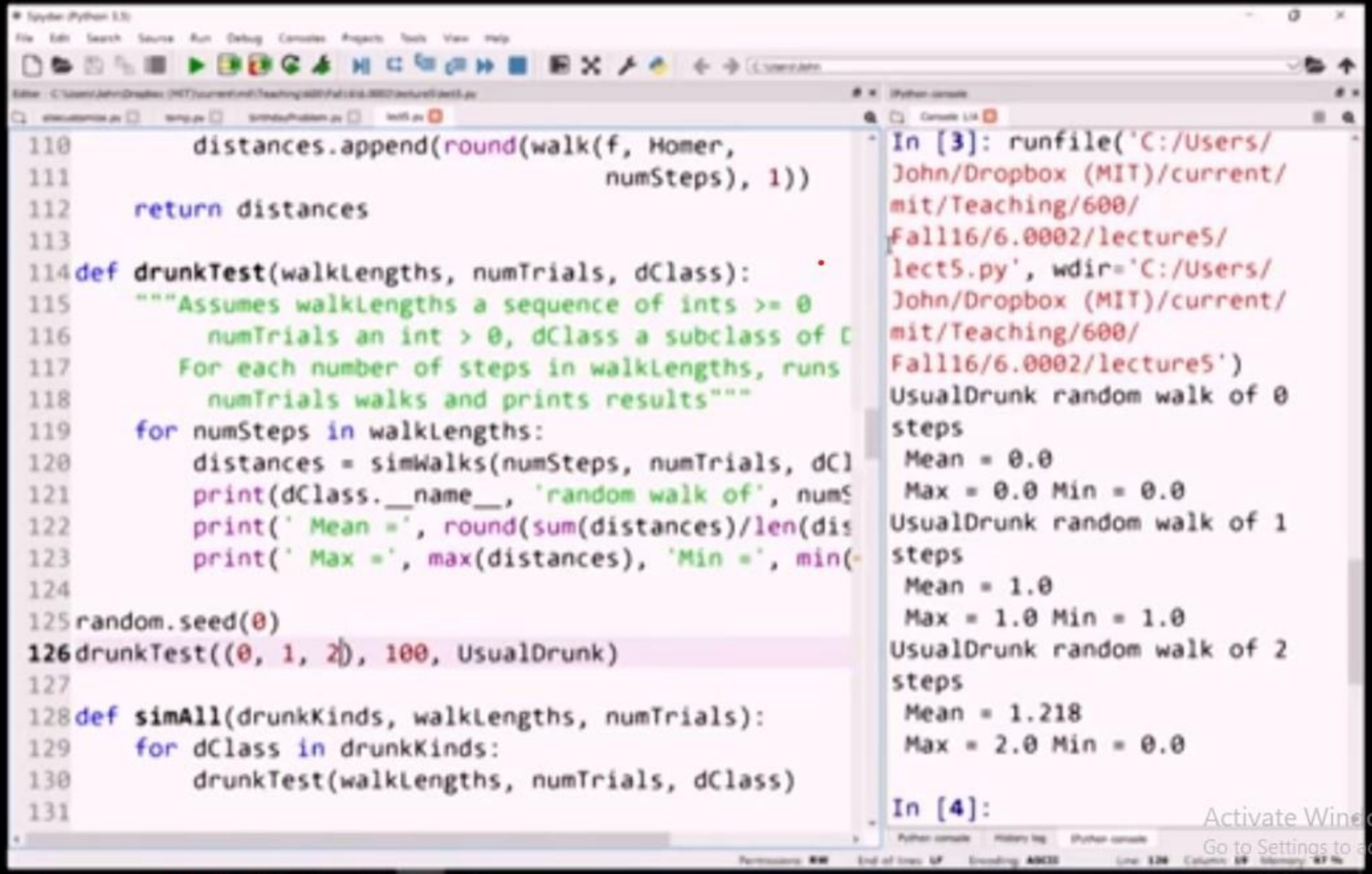
Plausible?

Sanity Check

```
drunkTest((0, 1, 2) 100, UsualDrunk)

UsualDrunk random walk of 0 steps
 Mean = 8.634
 Max = 21.6 Min = 1.4
UsualDrunk random walk of 1 steps
 Mean = 8.57
 Max = 22.0 Min = 0.0
UsualDrunk random walk of 2 steps
 Mean = 9.206
 Max = 21.6 Min = 1.4
```


```
distances.append(round(walk(f, Homer,
                            numTrials), 1))
```



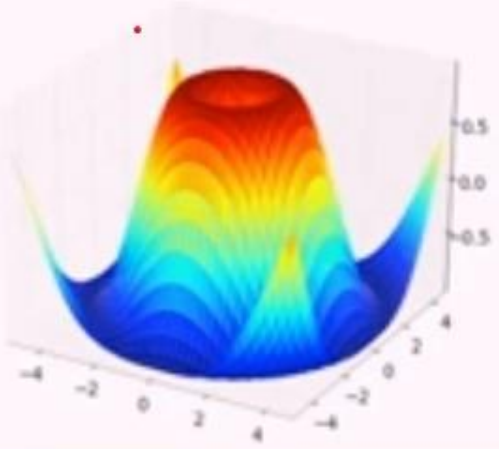

distances.append(round(walk(f, Homer,
numSteps), 1))
return distances
def drunkTest(walkLengths, numTrials, dClass):
"""Assumes walkLengths a sequence of ints >= 0
numTrials an int > 0, dClass a subclass of D
For each number of steps in walkLengths, runs
numTrials walks and prints results"""
for numSteps in walkLengths:
distances = simWalks(numSteps, numTrials, dCl
print(dClass.__name__, 'random walk of', numS
print(' Mean =', round(sum(distances)/len(dis
print(' Max =', max(distances), 'Min =', min(
random.seed(0)
drunkTest((0, 1, 2), 100, UsualDrunk)
def simAll(drunkKinds, walkLengths, numTrials):
for dClass in drunkKinds:
drunkTest(walkLengths, numTrials, dClass)
In [3]: runfile('C:/Users/
John/Dropbox (MIT)/current/
mit/Teaching/600/
Fall16/6.0002/lecture5/
lect5.py', wdir='C:/Users/
John/Dropbox (MIT)/current/
mit/Teaching/600/
Fall16/6.0002/lecture5')
UsualDrunk random walk of 0
steps
Mean = 0.0
Max = 0.0 Min = 0.0
UsualDrunk random walk of 1
steps
Mean = 1.0
Max = 1.0 Min = 1.0
UsualDrunk random walk of 2
steps
Mean = 1.218
Max = 2.0 Min = 0.0
In [4]:
Activate Windows
Go to Settings to activate Windows.

And the Masochistic Drunk?

```
random.seed(0)
simAll((UsualDrunk, MasochistDrunk),
       (1000, 10000), 100)
UsualDrunk random walk of 1000 steps
 Mean = 26.828
 Max = 66.3 Min = 4.2
UsualDrunk random walk of 10000 steps
 Mean = 90.073
 Max = 210.6 Min = 7.2
MasochistDrunk random walk of 1000 steps
 Mean = 58.425
 Max = 133.3 Min = 6.7
MasochistDrunk random walk of 10000 steps
 Mean = 515.575
 Max = 694.6 Min = 377.7
```

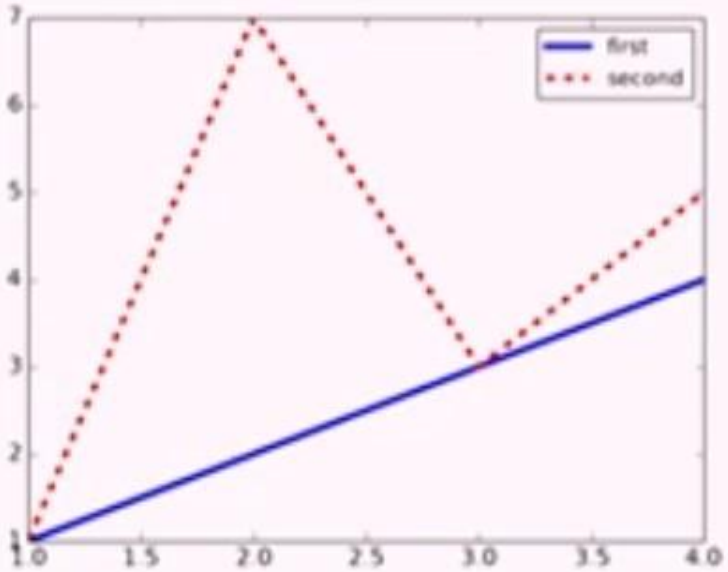
Pylab

- NumPy adds vectors, matrices, and many high-level mathematical functions
- SciPy adds mathematical classes and functions useful to scientists
- MatPlotLib adds an object-oriented API for plotting
- PyLab combines the other libraries to provide a MATLAB-like interface

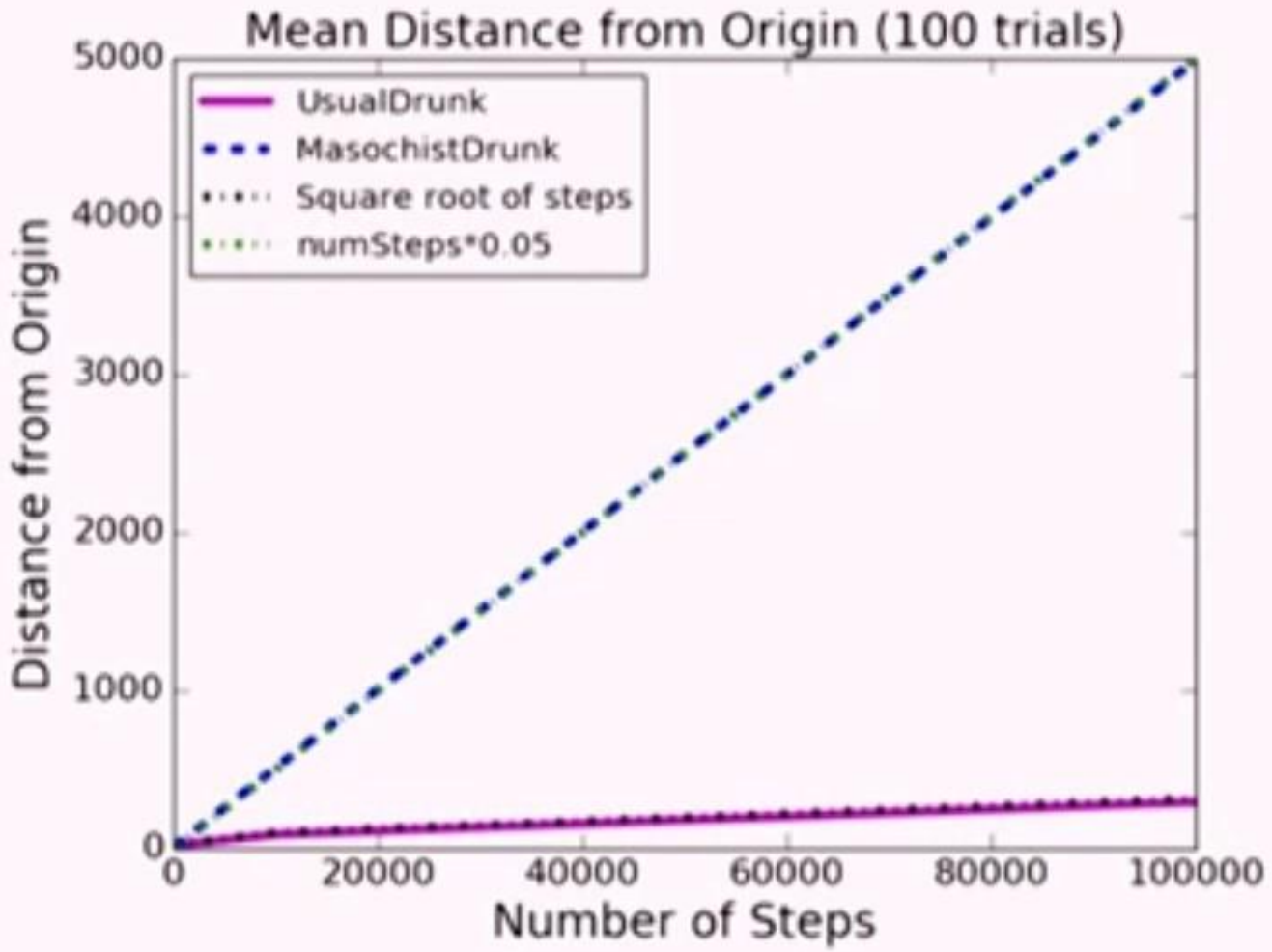


Example

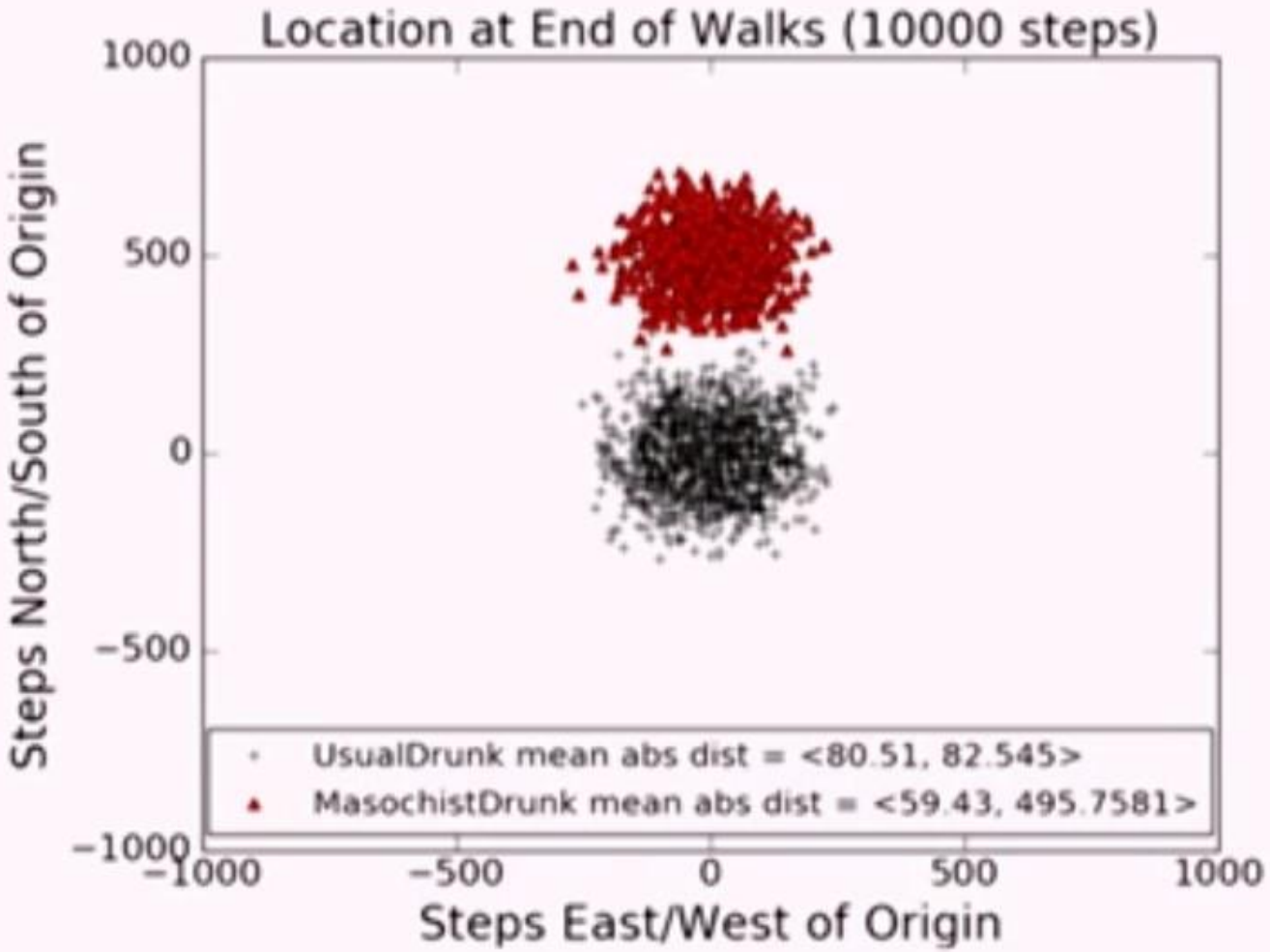
```
import pylab

xVals = [1, 2, 3, 4]
yVals1 = [1, 2, 3, 4]
pylab.plot(xVals, yVals1, 'b-', label = 'first')
yVals2 = [1, 7, 3, 5]
pylab.plot(xVals, yVals2, 'r--', label = 'second')
pylab.legend()
```



Distance Trends



Ending Locations



A Subclass of Field, part 1

```
class OddField(Field):
    def __init__(self, numHoles = 1000,
                 xRange = 100, yRange = 100):
        Field.__init__(self)
        self.wormholes = {}
        for w in range(numHoles):
            x = random.randint(-xRange, xRange)
            y = random.randint(-yRange, yRange)
            newX = random.randint(-xRange, xRange)
            newY = random.randint(-yRange, yRange)
            newLoc = Location(newX, newY)
            self.wormholes[(x, y)] = newLoc
```

A Subclass of Field, part 2

```
def moveDrunk(self, drunk):
    Field.moveDrunk(self, drunk)
    x = self.drunks[drunk].getX()
    y = self.drunks[drunk].getY()
    if (x, y) in self.wormholes:
        self.drunks[drunk] = self.wormholes[(x, y)]
```

Spots Reached During One Walk