def clear(n, clearProb, steps):
    numRemaining = [n]
    for t in range(steps):
        numRemaining.append(n*((1-clearProb)**t))
    pylab.plot(numRemaining, label = 'Exponential Decay')
    pylab.xlabel('Time')
    pylab.ylabel('Num Molecules')

def clearSim(n, clearProb, steps):
    numRemaining = [n]
    for t in range(steps):
        numLeft = numRemaining[-1]
        for m in range(numRemaining[-1]):
            if random.random() <= clearProb:
                numLeft -= 1
        numRemaining.append(numLeft)
    pylab.plot(numRemaining, 'r', label = 'Simulation')

def throwNeedles(numNeedles):
    inCircle = 0
    estimates = []
    for Needles in xrange(1, numNeedles + 1, 1):
        x = random.random()
        y = random.random()
        if (x*x + y*y)**0.5 <= 1.0:
            inCircle += 1
    return 4*(inCircle/float(Needles))

def getEst(numNeedles, numTrials = 20, toPrint = True):
    estimates = []
    for t in range(numTrials):
        piGuess = throwNeedles(numNeedles)
        estimates.append(piGuess)
        sDev = stdDev(estimates)
        curEst = sum(estimates)/len(estimates)
        if toPrint:
            print 'Est. = ' + str(curEst) + ', Std. dev. = ' + str(sDev)\
            + ', Needles = ' + str(numNeedles)
    return curEst, sDev

def estPi(precision, numTrials):
    numNeedles = 1000
    curStd = precision
    while curStd >= precision/2.0:
        curEst, curStd = getEst(numNeedles, numTrials)
        numNeedles *= 2
    return curEst, curStd

pi, std = estPi(0.005, 20)
print 'With 95% confidence, pi is between', pi-2*std, 'and', pi+2*std
def makeCurve(f, minVal, maxVal, numPts):
    xVals, yVals = [], []
    for i in range(numPts):
        xVal = random.uniform(minVal, maxVal)
        xVals.append(xVal)
        yVals.append(f(xVal))
    return xVals, yVals

def quadratic(x):
    return x**2

def makeCurve(f, minVal, maxVal, numPts):
    xVals, yVals = [], []
    for i in range(numPts):
        xVal = random.uniform(minVal, maxVal)
        xVals.append(xVal)
        yVals.append(f(xVal))
    return xVals, yVals

def quadratic(x):
    return x**2

xVals, yVals = makeCurve(quadratic, 0, 8, 1000)
pylab.plot(xVals, yVals, 'o')

def linear(x):
    return x

def integrate(a, b, f, numPts):
    ptSum = 0.0
    for p in range(numPts):
        ptSum += f(random.uniform(a, b))
    average = ptSum/numPts
    return average*(b - a)

print 'Integral of x from 0 to 8 =', integrate(0, 8, linear, 100000)
print 'Integral of x**2 from 0 to 8 =', integrate(0, 8, quadratic, 100000)

import math

def hard(x):
    if x > 2 and x < 6:
        return math.log(x**x, 2)
    else:
        return x + math.log(x**x, 2)