6.00 Handout, Lecture 16
(Not intended to make sense outside of lecture)

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

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 montyChoose(guessDoor, prizeDoor):
    if 1 != guessDoor and 1 != prizeDoor:
        return 1
    if 2 != guessDoor and 2 != prizeDoor:
        return 2
    return 3

def randomChoose(guessDoor, prizeDoor):
    if guessDoor == 1:
        return random.choice([2,3])
    if guessDoor == 2:
        return random.choice([1,3])
    return random.choice([1,2])

def simMontyHall(numTrials = 100, chooseFcn = montyChoose):
    stickWins = 0
    switchWins = 0
    noWin = 0
    prizeDoorChoices = [1, 2, 3]
    guessChoices = [1, 2, 3]
    for t in range(numTrials):
        prizeDoor = random.choice([1, 2, 3])
        guess = random.choice([1, 2, 3])
        toOpen = chooseFcn(guess, prizeDoor)
        if toOpen == prizeDoor:
            noWin += 1
        elif guess == prizeDoor:
            stickWins += 1
        else:
            switchWins += 1
    return (stickWins, switchWins)
```

```
def displayMHSim(simResults):
    stickWins, switchWins = simResults
    pylab.pie([stickWins, switchWins], colors = ['r', 'g'],
              labels = ['stick', 'change'], autopct = '%.2f%%')
    pylab.title('To Switch or Not to Switch')

def stdDev(X):
    mean = sum(X)/float(len(X))
    tot = 0.0
    for x in X:
        tot += (x – mean)**2
    return (tot/len(X))**0.5

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 estPi(precision = 0.01, numTrials = 20):
    numNeedles = 1000
    numTrials = 20
    sDev = precision
    while sDev >= (precision/4.0):
        estimates = []
        for t in range(numTrials):
            piGuess = throwNeedles(numNeedles)
            estimates.append(piGuess)
        sDev = stdDev(estimates)
        curEst = sum(estimates)/len(estimates)
        curEst = sum(estimates)/len(estimates)
        print 'Est. = ' + str(curEst) +\
              ', Std. dev. = ' + str(sDev)\
              + ', Needles = ' + str(numNeedles)
        numNeedles *= 2
    return curEst

def integrate(a, b, f, numPins):
    pinSum = 0.0
    for pin in range(numPins):
        pinSum += f(random.uniform(a, b))
    average = pinSum/numPins
    return average*(b - a)

def doubleIntegrate(a, b, c, d, f, numPins):
    pinSum = 0.0
    for pin in range(numPins):
        x = random.uniform(a, b)
        y = random.uniform(c, d)
```

```
    pinSum += f(x, y)
average = pinSum/numPins
return average*(b - a)*(d - c)
```

6.00SC Introduction to Computer Science and Programming
Spring 2011