6.00 Handout, Lecture 18
(Not intended to make sense outside of lecture)

```
def rSquare(measured, estimated):
    """measured: one dimensional array of measured values
       estimate: one dimensional array of predicted values"""
    EE = ((estimated - measured)**2).sum()
    mMean = measured.sum()/float(len(measured))
    MV = ((mMean - measured)**2).sum()
    return 1 - EE/MV

def getXSpeed(a, b, c, minX, maxX):
    """minX and maxX are distances in inches"""
    xMid = (maxX - minX)/2.0
    yPeak = a*xMid**2 + b*xMid + c
    g = 32.16*12 #accel. of gravity in inches/sec/sec
    t = (2.0*yPeak/g)**0.5
    return xMid/(t*12.0)

def processTrajectories(fName):
    distances, heights = getTrajectoryData(fName)
    distances = pylab.array(distances)*36
    totHeights = pylab.array([0]*len(distances))
    for h in heights:
        totHeights = totHeights + pylab.array(h)
    pylab.title('Trajectory of Projectile (Mean of 4 Trials)')
    pylab.xlabel('Inches from Launch Point')
    pylab.ylabel('Inches Above Launch Point')
    meanHeights = totHeights/len(heights)
    pylab.plot(distances, meanHeights, 'bo')
    a,b,c = pylab.polyfit(distances, meanHeights, 2)
    altitudes = a*(distances**2) +  b*distances + c
    speed = getXSpeed(a, b, c, distances[-1], distances[0])
    pylab.plot(distances, altitudes, 'g',
               label = 'Quad. Fit' + ', R2 = '
               + str(round(rSquare(meanHeights, altitudes), 2))
               + ', Speed = ' + str(round(speed, 2)) + 'feet/sec')
    pylab.legend()
```

	Value	Weight	Value/Weight
Clock	175	10	17.5
Painting	90	9	10
Radio	20	4	5
Vase	50	2	25
Book	10	1	10
Computer	200	20	10

```
class Item(object):
    def __init__(self, n, v, w):
        self.name = n
        self.value = float(v)
        self.weight = float(w)
    def getName(self):
        return self.name
    def getValue(self):
        return self.value
    def getWeight(self):
        return self.weight
    def __str__(self):
        result = '<' + self.name + ', '
                 + str(self.value) + ', '\
                 + str(self.weight) + '>'
        return result

def buildItems():
    names = ['clock', 'painting', 'radio', 'vase', 'book', 'computer']
    vals = [175,90,20,50,10,200]
    weights = [10,9,4,2,1,20]
    Items = []
    for i in range(len(vals)):
        Items.append(Item(names[i], vals[i], weights[i]))
    return Items

def greedy(Items, maxWeight, keyFcn):
    assert type(Items) == list and maxWeight >= 0
    ItemsCopy = sorted(Items, key=keyFcn, reverse = True)
    result = []
    totalVal = 0.0
    totalWeight = 0.0
    i = 0
    while totalWeight < maxWeight and i < len(Items):
        if (totalWeight + ItemsCopy[i].getWeight()) <= maxWeight:
            result.append((ItemsCopy[i]))
            totalWeight += ItemsCopy[i].getWeight()
            totalVal += ItemsCopy[i].getValue()
        i += 1
    return (result, totalVal)

def value(item):
    return item.getValue()

def weightInverse(item):
    return 1.0/item.getWeight()

def density(item):
    return item.getValue()/item.getWeight()

def testGreedy(Items, constraint, getKey):
    taken, val = greedy(Items, constraint, getKey)
    print ('Total value of items taken = ' + str(val))
    for item in taken: print '  ', item
```

6.00SC Introduction to Computer Science and Programming
Spring 2011