

**6.00 Handout, Lecture 17**  
**(Not intended to make sense outside of lecture)**

	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

a	b	c	d	combos
0	0	0	0	{}
0	0	0	1	{d}
0	0	1	0	{c}
0	0	1	1	{c,d}
0	1	0	0	{b}
0	1	0	1	{b,d}
0	1	1	0	{b,c}
0	1	1	1	{b,c,e}
1	0	0	0	{a}
1	0	0	1	{a,d}
1	0	1	0	{a,c}
1	0	1	1	{a,c,d}
1	1	0	0	{a,b}
1	1	0	1	{a,b,d}
1	1	1	0	{a,b,c}
1	1	1	1	{a,b,c,d}

```

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

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

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

```

```

def testGreedy():
    Items = buildItems()
    print('Items to choose from:')
    for item in Items:
        print(item)
    print 'Use greedy by value to fill a knapsack of size 20'
    testGreedy(Items, 20, Item.getValue)
    print 'Use greedy by weight to fill a knapsack of size 20'
    testGreedy(Items, 20, weightInverse)
    print 'Use greedy by density to fill a knapsack of size 20'
    testGreedy(Items, 20, density)

def dToB(n, numDigits):
    """requires: n is a natural number less than 2**numDigits
       returns a binary string of length numDigits representing the
           the decimal number n."""
    assert type(n)==int and type(numDigits)==int and n >=0 and n < 2**numDigits
    bStr = ''
    while n > 0:
        bStr = str(n % 2) + bStr
        n = n//2
    while numDigits - len(bStr) > 0:
        bStr = '0' + bStr
    return bStr

def genPset(Items):
    """Generate a list of lists representing the power set of Items"""
    numSubsets = 2**len(Items)
    templateLen = len(Items)
    templates = []
    for i in range(numSubsets):
        templates.append(dToB(i, templateLen))
    pset = []
    for t in templates:
        elem = []
        for j in range(len(t)):
            if t[j] == '1':
                elem.append(Items[j])
        pset.append(elem)
    return pset

def chooseBest(pset, constraint, getVal, getWeight):
    bestVal = 0.0
    bestSet = None
    for Items in pset:
        ItemsVal = 0.0
        ItemsWeight = 0.0
        for item in Items:
            ItemsVal += getVal(item)
            ItemsWeight += getWeight(item)
        if ItemsWeight <= constraint and ItemsVal > bestVal:
            bestVal = ItemsVal
            bestSet = Items
    return (bestSet, bestVal)

```