In [1]:
```
class Food(object):
    def __init__(self, n, v, w):
        self.name = n
        self.value = v
        self.calories = w
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) +
```

In [22]:
```
def buildMenu(names, values, calories):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i]))
    return menu
```

In [23]:
```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

In [24]:
```
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)
```

In [5]:
```
def testGreedys(foods, maxUnits):
    print('Use greedy by value to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, lambda x: 1/Food.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.density)
```

Task 1: Change the maxUnits to 100

In [55]:
```
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', '
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
foods = buildMenu(names, values, calories)
testGreedys(foods, 100)
```

```
Use greedy by value to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95>

Use greedy by cost to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95>

Use greedy by density to allocate 100 calories
Total value of items taken = 50.0
    apple: <50, 95>
```

Task 2: Modify codes to add additional weight (criterion) to select food items.

In [ ]:
```
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', '
values = [89,90,95,100,90,79,50,10,60]
calories = [123,154,258,354,365,150,95,195,155]
foods = buildMenu(names, values, calories)
testGreedys(foods, 1900)
```

Task 3: Test your modified code to test the greedy algorithm to select food items with your additional weight.

In [63]:
```
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', '
values = [89,90,95,100,90,79,50,10,60]
calories = [123,154,258,354,365,150,95,195,155]
foods = buildMenu(names, values, calories)
testGreedys(foods, 1900)
```

```
Use greedy by value to allocate 1900 calories
Total value of items taken = 663.0
    burger: <100, 354>
    pizza: <95, 258>
    beer: <90, 154>
    fries: <90, 365>
    wine: <89, 123>
    cola: <79, 150>
    cake: <60, 155>
    apple: <50, 95>
    donut: <10, 195>

Use greedy by cost to allocate 1900 calories
Total value of items taken = 663.0
    apple: <50, 95>
    wine: <89, 123>
    cola: <79, 150>
    beer: <90, 154>
    cake: <60, 155>
    donut: <10, 195>
    pizza: <95, 258>
    burger: <100, 354>
    fries: <90, 365>

Use greedy by density to allocate 1900 calories
Total value of items taken = 663.0
    wine: <89, 123>
    beer: <90, 154>
    cola: <79, 150>
    apple: <50, 95>
    cake: <60, 155>
    pizza: <95, 258>
    burger: <100, 354>
    fries: <90, 365>
    donut: <10, 195>
```

Supplementary Activity

Problem: You are starting your road to becoming fit because it your new years resolution. One of your goals is to minimize your caloric intake and you go outside and go to your favorite food mall to eat. You see all of your favorite foods in the menu. However, You remember that you are bounded by the amount of calories to 650 calories. Which among the among the foods are you going to buy so that you will meet the 650 calory mark?

Greedy Algorithm

In [ ]:
```
class Food(object):
    def __init__(self, n, v, w):
        self.name = n
        self.value = v
        self.calories = w
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)+ ', ' + str(self.calories) +
```

In [ ]:
```
def buildMenu(names, values, calories):
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],calories[i]))
    return menu
```

In [ ]:
```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,         keyFunction maps elements of
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

In [ ]:
```
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)
```

In [ ]:
```
def testGreedys(foods, maxUnits):
    print('Use greedy by value to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.getValue)
    print('\nUse greedy by cost to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, lambda x: 1/Food.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,          'calories')
    testGreedy(foods, maxUnits, Food.density)
```

In [51]:
```
names = ['shawarma rice', 'longsilog', 'burger','5 piece calamares', 'sizzling
values = [55,65,35,25,85, 45]
calories = [150,200, 320,130,390,210]
foods = buildMenu(names, values, calories)
testGreedys(foods, 650)
```

```
Use greedy by value to allocate 650 calories
Total value of items taken = 150.0
    sizzling sisig: <85, 390>
    longsilog: <65, 200>

Use greedy by cost to allocate 650 calories
Total value of items taken = 145.0
    5 piece calamares: <25, 130>
    shawarma rice: <55, 150>
    longsilog: <65, 200>

Use greedy by density to allocate 650 calories
Total value of items taken = 165.0
    shawarma rice: <55, 150>
    longsilog: <65, 200>
    siomai rice: <45, 210>
```

Brute Force Algorithm

In [30]:
```
def maxVal(toConsider, avail):
    """Assumes toConsider a list of items, avail a weight
       Returns a tuple of the total value of a solution to the
         0/1 knapsack problem and the items of that solution"""
    if toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getCost() > avail:
        #Explore right branch only
        result = maxVal(toConsider[1:], avail)
    else:
        nextItem = toConsider[0]
        #Explore left branch
        withVal, withToTake = maxVal(toConsider[1:],
                                     avail - nextItem.getCost())
        withVal += nextItem.getValue()
        #Explore right branch
        withoutVal, withoutToTake = maxVal(toConsider[1:], avail)
        #Choose better branch
        if withVal > withoutVal:
            result = (withVal, withToTake + (nextItem,))
        else:
            result = (withoutVal, withoutToTake)
    return result
```

In [41]:
```
def testMaxVal(foods, maxUnits, printItems = True):
    print('Use search tree to allocate', maxUnits,
          'calories')
    val, taken = maxVal(foods, maxUnits)
    print('Total costs of foods taken =', val)
    if printItems:
        for item in taken:
            print('   ', item)
```

In [49]:
```
names = ['shawarma rice', 'longsilog', 'burger','5 piece calamares', 'sizzling
values = [55,65,35,25,85, 45]
calories = [265,200,320,130,390,210]
foods = buildMenu(names, values, calories)
testMaxVal(foods, 650)
```

```
Use search tree to allocate 650 calories
Total costs of foods taken = 150
    sizzling sisig: <85, 390>
    longsilog: <65, 200>
```

Conclusion

From this hands on activity, we experimented on how both the greedy algorithm and brute force algorithm work. From what I understood is that the greedy algorithm picks the best option in the list depending on the attributes that are specified by the programmer whether it is by price, value, and etc. From the supplementary activity I was able to make a constraint of 650 calories and upon running the program, the program showed me the best choice by value and calories, and density. It shows there that Longsilog is among the foods that is one of the best choice to buy. Moreover, what the brute force algorithm does is that it goes through all the list of items to find the best possible choice. However, if it runs through a long list of items its time complexity is slower. In the supplementary activity, after the program runs through the list of food items, it showed that sizzling sisig and longsilog were the two foods that is the best possible choice.