```
#Create the methods inside the Food Class:

class Food(object):
  def __init__(self, n, v, w, p):
    self.name = n
    self.value = v
    self.calories = w
    self.prefer = p #Preference (Small/Large/Medium)

  def getValue(self):
    return self.value
  def getCost(self):
    return self.calories
  def getPref(self):
    return self.prefer
  def density(self):
    return self.getValue()/self.getCost()
  def __str__(self):
    return self.name + ': < ' + str(self.value) + ', ' + str(self.calories) + ' >'

#Create a build menu

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

#Create a method greedy to return total value and cost of added food based on the desi
def greedy(items, maxCost, keyFunction):
    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])
            result.append(itemsCopy[i].getPref())
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)


def testGreedy(items, constraint, keyFunction, ):
  taken, val = greedy(items, constraint, keyFunction)
  print('Total Values of items taken =' , val)
  for item in taken:
    print(' ', item)

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)

  #Create arrays of food name, values, and calories

names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
preference = ["S", "L", "M", "L", "S", "L", "L", "M", "M"]
foods = buildMenu(names, values, calories, preference)
testGreedys(foods, 2000)
```

```
Use greedy by value to allocate 2000 calories
Total Values of items taken = 603.0
  burger: < 100, 354 >
  L
  pizza: < 95, 258 >
  M
  beer: < 90, 154 >
  L
  fries: < 90, 365 >
  S
  wine: < 89, 123 >
  S
  cola: < 79, 150 >
  L
  apple: < 50, 95 >
  L
  donut: < 10, 195 >
  M

Use greedy by  cost to allocate 2000 calories
Total Values of items taken = 603.0
  apple: < 50, 95 >
  L
  wine: < 89, 123 >
  S
  cola: < 79, 150 >
  L
  beer: < 90, 154 >
  L
  donut: < 10, 195 >
  M
  pizza: < 95, 258 >
  M
  burger: < 100, 354 >
  L
  fries: < 90, 365 >
  S
```

```
Use greedy by density to allocate 2000 calories
Total Values of items taken = 603.0
  wine: < 89, 123 >
  S
  beer: < 90, 154 >
  L
  cola: < 79, 150 >
  L
  apple: < 50, 95 >
  L
  pizza: < 95, 258 >
  M
  burger: < 100, 354 >
  L
  fries: < 90, 365 >
  S
  donut: < 10, 195 >
  M
```

Task 1: Change the maxUnits to 100

```
# Code for Task 1
testGreedys(foods, 100)
```

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

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

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

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

```
# For task two. I added a preference. where p <item>.
# L stands for Large, M stands for Medium, S for small
# Example: Large <Burger>, Small <Fries>, etc



#Create the methods inside the Food Class:

class Food(object):
  def __init__(self, n, v, w, p):
    self.name = n
    self.value = v
```

```
    self.calories = w
    self.prefer = p #Preference (Small/Large/Medium)

  def getValue(self):
    return self.value
  def getCost(self):
    return self.calories
  def getPref(self):
    return self.prefer
  def density(self):
    return self.getValue()/self.getCost()
  def __str__(self):
    return self.name + ': < ' + str(self.value) + ', ' + str(self.calories) + ' >'

#Create a build menu

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

#Create a method greedy to return total value and cost of added food based on the desi
def greedy(items, maxCost, keyFunction):
    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])
            result.append(itemsCopy[i].getPref())
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)


def testGreedy(items, constraint, keyFunction):
  taken, val = greedy(items, constraint, keyFunction)
  print('Total Values of items taken =' , val)
  for item in taken:
    print(' ', item)

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)

  #Create arrays of food name, values, and calories
```

```
names = ['wine', 'beer', 'pizza', 'burger', 'fries','cola', 'apple', 'donut', 'cake']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]
preference = ["S", "L", "M", "L", "S", "L", "L", "M", "M"]
foods = buildMenu(names, values, calories, preference)
testGreedys(foods, 2000)
```

```
Use greedy by value to allocate 2000 calories
Total Values of items taken = 603.0
  burger: < 100, 354 >
  L
  pizza: < 95, 258 >
  M
  beer: < 90, 154 >
  L
  fries: < 90, 365 >
  S
  wine: < 89, 123 >
  S
  cola: < 79, 150 >
  L
  apple: < 50, 95 >
  L
  donut: < 10, 195 >
  M

Use greedy by  cost to allocate 2000 calories
Total Values of items taken = 603.0
  apple: < 50, 95 >
  L
  wine: < 89, 123 >
  S
  cola: < 79, 150 >
  L
  beer: < 90, 154 >
  L
  donut: < 10, 195 >
  M
  pizza: < 95, 258 >
  M
  burger: < 100, 354 >
  L
  fries: < 90, 365 >
  S

Use greedy by density to allocate 2000 calories
Total Values of items taken = 603.0
  wine: < 89, 123 >
  S
  beer: < 90, 154 >
  L
  cola: < 79, 150 >
  L
```

```
apple: < 50, 95 >
L
pizza: < 95, 258 >
M
burger: < 100, 354 >
L
fries: < 90, 365 >
S
donut: < 10, 195 >
M
```

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

```
# RESULTS:
# Use greedy by value to allocate 2000 calories
# Total Values of items taken = 603.0
#   burger: < 100, 354 >
#   L
#   pizza: < 95, 258 >
#   M
#   beer: < 90, 154 >
#   L
#   fries: < 90, 365 >
#   S
#   wine: < 89, 123 >
#   S
#   cola: < 79, 150 >
#   L
#   apple: < 50, 95 >
#   L
#   donut: < 10, 195 >
#   M

# Use greedy by  cost to allocate 2000 calories
# Total Values of items taken = 603.0
#   apple: < 50, 95 >
#   L
#   wine: < 89, 123 >
#   S
#   cola: < 79, 150 >
#   L
#   beer: < 90, 154 >
#   L
#   donut: < 10, 195 >
#   M
#   pizza: < 95, 258 >
#   M
#   burger: < 100, 354 >
#   L
```

```
#   fries: < 90, 365 >
#   S

# Use greedy by density to allocate 2000 calories
# Total Values of items taken = 603.0
#   wine: < 89, 123 >
#   S
#   beer: < 90, 154 >
#   L
#   cola: < 79, 150 >
#   L
#   apple: < 50, 95 >
#   L
#   pizza: < 95, 258 >
#   M
#   burger: < 100, 354 >
#   L
#   fries: < 90, 365 >
#   S
#   donut: < 10, 195 >
#   M
```

SUPPLEMENTARY ACTIVITY

```
#Airport Baggage Problem:

class item(object):
  def __init__(self, name, cost,weight):
    self.name = name
    self.val = cost
    self.weight = weight

  def getWeight(self):
    return self.weight
  def getCost(self):
    return self.val
def itemList(names, cost ,weight):
  empty = []
  for i in range(len(weight)):
    empty.append((item(names[i], cost[i] ,weight[i])))
  return empty

def greedy(obj, maxWeight, keyFunction):
  emptyCopy = sorted(obj, key = keyFunction, reverse = True)

  result = []
  totalWeight = 0.0
  for i in range(len(emptyCopy)):
    if (totalWeight + emptyCopy[i].getWeight()) <= maxWeight:
```

```
      result.append(emptyCopy[i])
      totalCost += emptyCopy[i].getCost()
      totalWeight += emptyCopy[i].getWeight()
    return(result, totalWeight)

def test(obj, constraint, keyFunction):
  a, w = greedy(obj, constraint, keyFunction)
  print("Total weight of the bag is ", w)
  for item in a:
    print("-->", item)

def tryAlgo(i, maxWeight):
  print('Use greedy by weight to allocate', maxWeight)
  test(i, maxWeight, item.getWeight)


name = ["Toothbrush", "Pillow", "Clothes", "Shoes", "Gadgets","Snacks", "Container"]
values = [89,90,95,100,90,79]
weights = [10, 50, 30, 100, 500, 40]
i = itemList(name,values,weights)
tryAlgo(i, 150)
```

```
Use greedy by weight to allocate 150
Total weight of the bag is  0.0
```

```
#Brute force
```

Conclusion:

For the supplementary