```
# /Users/Kamyar/Box Sync/Computer Science/Term project/TermProjectFinalMath.py
#Name:Kamyar Ghiam
#Project: Deciduo
#15-112 -- Fall 2017

#Food Database
import Databases.Final_Databases.FoodDatabase as FOOD
#Exercise Database
import Databases.Final_Databases.Exercise_Database_Final as EXERCISE
#Used for Linear Equations. This is a module from the scipy.optimize package
from scipy.optimize import brenth
import decimal, string
#this is to reload the databases
import importlib as imp
#taken from course website
def roundHalfUp(d):
    # Round to nearest with ties going away from zero.
    rounding = decimal.ROUND_HALF_UP
    return int(decimal.Decimal(d).to_integral_value(rounding=rounding))
```

```
####################################
# Food
####################################

class Food(object):
    #module
    def __init__(self, discrete = None, healthy = [], happy = [],\
    calories = [], money = [], Wcalories = [], Wmoney = [], howhealthy = [],\
    howhappy = [], alpha = [], anotherinput =  True, countvariables = 0,\
    max = 2, min  = 1):
        self.max = max #maximum food items you can ask for
        self.min = min
        self.discrete = discrete #checks if the food is divisible
        #Fixed Judgement
        self.healthy = healthy
        self.happy = happy
        #Costs
        self.calories = calories
        self.money = money
        #Constraints (W = willing to eat or pay)
        self.Wcalories =
Wcalories
        self.Wmoney = Wmoney
        #Variables judgement
        self.howhealthy = howhealthy
        self.howhappy = howhappy
        #Algorithm
        self.alpha = alpha
        #Checks if user wants to input another food
        self.anotherinput = anotherinput
        #Counts how many food items were added
        self.countvariables = countvariables
        #collects food items
        self.foods = []
        #this is for the machine learning algorithm
        self.learn = 1.5
    #solves for lambda and then can be used to get x and y values
    def lambdaSolverMoneyConstraint(self):
        epsilon = 10**-8
        delta = 10**8
        #calculated function of legranage optimization (derviation can be shown)
        def function(x):
            a1 = self.alpha[0]
```

```
#alpha values
            a2 = self.alpha[1]
            m1 = self.money[0] #money constraint
            m2 = self.money[1]
            return (m1)*(((x*m1)/(a1))**(1/(a1-1))+(m2)*(((x*m2)/(a2))**(1/(a2-1))) - self.Wmoney[0]
        return brenth(function, epsilon, delta)
    def getValuesFromLambdaMoney(self):
        L = self.lambdaSolverMoneyConstraint() #lambda
        #these are calculated values
        a1 =  self.alpha[0]
        a2 = self.alpha[1]
        m1 = self.money[0] #money constraint
        m2 = self.money[1]
        first_food = ((L*m1)/(a1))**(1/(a1-1))
        second_food = ((L*m2)/(a2))**(1/(a2-1))
        return [first_food, second_food]
    def lambdaSolverCalorieConstraint(self):
        epsilon = 10**-8
        delta = 10**8
    #calculated function of legranage optimization (derviation can be shown)
        def function(x):
            a1 = self.alpha[0] #alpha values
            a2 = self.alpha[1]
            c1 = self.calories[0] #calorie constraint
            c2 = self.calories[1]
            return (c1)*(((x*c1)/(a1)))**(1/(a1-1))+(c2)*(((x*c2)/(a2))**(1/(a2-1))) - self.Wcalories[0]
        return brenth(function, epsilon, delta)

    def getValuesFromLambdaCalorie(self):
        L = self.lambdaSolverCalorieConstraint() #lambda
        #these are calculated values
        a1 =  self.alpha[0]
        a2 = self.alpha[1]
        c1 = self.calories[0] #calorie constraint
        c2 = self.calories[1]
        first_food = ((L*c1)/(a1))**(1/(a1-1))
        second_food =
((L*c2)/(a2))**(1/(a2-1))
        return [first_food, second_food]
    def collectInitialself(self):
        #Constraints (W = willing to eat or pay)
        (self.Wcalories).append(int(input("How many calories are you willing to eat? ")))
        (self.Wmoney).append(int(input("How much money are you willing to spend in total? ")))
        #Variables judgement
        (self.howhealthy).append(\
        int(input("How important is it for you to be healthy in this meal?(From 1-10) ")))
        (self.howhappy).append(int(input("How important is it for you to feel satisfied from this meal?(from 1-10) ")))
        discrete = input("Is your food divisble? (i.e. do you want some of each food?) ")
        if discrete == "Yes":
            self.discrete = False
```

```python
        else: self.discrete = True

    def checkDictioanryOfFood(self, search):
        search = search.lower()
        results = []
        for food in FOOD.dictionary_of_food:
            if search in food.lower():
                results.append([food, FOOD.dictionary_of_food[food]])
        if results == []:
            return False
        return results

    def collectFood(self):
        self.countvariables += 1
        #Fixed Judgement
        print()
        print("Food item %d:" % self.countvariables)
        print()
        food = input("What is the name of your food? ")
        results = self.checkDictioanryOfFood(food)
        if results == False:
            print("Sorry, we don't have that food. Please enter
it manually (we will store this
entry for next time). ")
self.newFoodItem(food)
         else:
             print("Which do you
want?" )
             print()
             print(results)
         #####PRINT POSSIBLE
FOOD ITEMS AND ASK WHICH THEY WANT
         ###THEN, APPEND THEIR
ANSWER TO THE DATA
         ###LATER, YOU NEED TO
BE ABLE TO EDIT DICTIONARY
             print()
             result = results[0]
#CHANGE THIS
self.foods.append(results[0])#CHANGE
THIS
             print("Okay, you
selected %s" % result[0])#CHANGE
THIS
(self.healthy).append(result[1][3])
(self.calories).append(result[1][0])

         print()
         #variable judgemnet
(self.happy).append(int(input(\
        "How happy would it make you to eat one serving?(From 1-10) ")))
        (self.money).append(int(input("How much does it cost per serving (In dollars)? ")))
        if 1 < self.countvariables < self.max-1:
            anotherinput = input("Do you want to add another food item?(Yes or No) ")
            if anotherinput == "No":
                self.anotherinput = False

    def newFoodItem(self,food):
        name = food
        healthy = int(input("How healthy is this food item?(From 1-10) "))
        (self.healthy).append(healthy)
        #Costs
        calories = int(input("How many calories is one serving? "))
        (self.calories).append(calories)
        servings = input("How much is one serving? (for example, write 1 oz) ")
        serving_number = 0
        serving_word = ""
        for letter in range(len(servings)):
            if servings[letter] in string.ascii_letters:
                serving_number = int(servings[0:letter-1])
                serving_word = servings[letter:]
                break
        food_entry = [name, (calories, serving_word, serving_number,healthy)]
        self.foods.append(food_entry)
        Food.addToFoodDictionary(food_entry)
        print(self.foods)

    @staticmethod
    def addToFoodDictionary(food):
        #finds where to add it in the dictionary
        with open("Databases/Final_Databases/Food Database.py") as file:
            data = file.readlines()
        dictionary = data[1]
        index = data[1].index("{")
        #creates entry
        entryName = food[0]
        entryTuple = food[1]
        dictionary = data[0]+ dictionary[:index+1] + "'%s':%s," % \
        (str(entryName), str(entryTuple))+ dictionary[index+1:]
        #writes newdictionary
        file = open("Databases/Final_Databases/Food Database.py", "w")
        file.write(dictionary)

        file.close()
        #reloads dictionary
        imp.reload(FOOD)

    def getAlpha(self):
        h1 = self.howhealthy[0]
        h2 = self.howhappy[0]
        t1x = self.healthy[0]
        t1y = self.healthy[1]
        t2x = self.happy[0]
        t2y = self.happy[1]
        alpha1 = 1/(self.learn*(h1*t1y + h2*t2y)) #This is a calculated values based on a formula I created
        alpha2 = 1/(h1*t1x + h2*t2x)

```

```
(self.alpha).append(alpha1)
(self.alpha).append(alpha2)
    def compareUtilityFunction(self):
        #finds the smaller utility inside constraints
        def utilityFunction(x, y):
            return x**(self.alpha[0]) + y**(self.alpha[1])
        x1 = self.money_solution[0]
        y1 = self.money_solution[1]
        x2 = self.calorie_solution[0]
        y2 = self.calorie_solution[1]
        if utilityFunction(x1, y1) <= utilityFunction(x2, y2):
            answer = [x1,y1]
        else: answer = [x2,y2]
        #finds the better food
        if answer[0] >= answer[1]:
            better_name = self.foods[0][0]
        else:
            better_name = self.foods[1][0]
        print(answer)
        if self.discrete == True:
            return "Eating %s is the better choice" % better_name
        else:
            servings1 = answer[0]*self.foods[0][1][2]
            servings2 = answer[1]*self.foods[1][1][2]
            food1 = self.foods[0][0]
            food2 = self.foods[1][0]
            servingword1 = self.foods[0][1][1]
            servingword2 = self.foods[1][1][1]
            return "For %s, I would have %0.2f (serving %s), and for %s, I would have %0.2f (serving %s)"\
        % (food1, servings1,servingword1, food2, servings2, servingword2)
        """
    #machine learning feature
    def adjustAlgorithm(self):
        """
#################

def runFood():
    food = Food()
    food.collectInitialself()
    food.collectFood()
    while food.anotherinput and food.countvariables < food.max:
        food.collectFood()
    food.getAlpha()
    food.money_solution = food.getValuesFromLambdaMoney()
    food.calorie_solution = food.getValuesFromLambdaCalorie()
    print(food.compareUtilityFunction())

##############









####################################
# Exercise
####################################

#could not use inherticance because the optimization algorithm is different
class Exercise(object):
    def __init__(self, max = 5, min = 1, discrete = None, happy = [],\
    calpermin = [], Wtime = [], Mcalories = [], howhappy = [], alpha = [],\
    anotherinput =  True, countvariables = 0, age = 0, weight = 0):
        #module
        self.max = max #maximum exercise items you can ask for
        self.min =  min
        self.discrete = discrete #checks if you can do multiple exercises
        #Fixed Judgement
        self.happy = happy
        self.calpermin = calpermin
        #Constraints (W = willing time. M is minimum calories wanted to be burned)
        self.Wtime = Wtime
        self.Mcalories = Mcalories
        #Variables judgement
        self.howhappy = howhappy
        #Algorithm
        self.alpha = alpha
        self.anotherinput = anotherinput #Checks if user wants to input another exercise
        self.countvariables = countvariables #Counts how many
```

```
exercise items were added
        self.age = age
        self.weight = weight
        self.exerciseNames = []

    def getAlpha(self):
        maximum = 11 #I chose 11 because I don't want to get a divide by 0 error when subtracting maximum (10)
        h = (maximum - self.howhappy[0])
        for i in range(len(self.happy)):
            h1 = maximum - self.happy[i]
            alpha = 1/(2*h*h1) #calculated values based on a formula I created
            (self.alpha).append(alpha)


    def collectInitialself(self):
        #Constraints (W = willing to eat or pay)
        (self.weight) = (int(input("How much do you weigh? ")))
        (self.age) = (int(input("How old are you? ")))

        (self.Mcalories).append(int(input("How many calories do you want to burn? ")))
        #Variables judgement
        (self.howhappy).append(\
        int(input("How important is it for you to enjoy this workout?(From 1-10) ")))
        discrete = input("Do you want to do multiple workouts?(i.e. split your time between workouts?) ")
        if discrete == "Yes":
            self.discrete = False
        else: self.discrete = True

    def collectExercise(self):
        minutes = 60
        self.countvariables += 1
        #Fixed Judgement
        print()
        print("Exercise item %d:" % self.countvariables)
        print()
        exercise = input("What is the name of your exercise? ")
        results = self.checkDictioanryOfExercise(exercise)
        if results == False:
            print("Sorry, we don't have that exercise. Please enter it manually.")
            self.newExerciseItem(exercise)
        else:
            self.exerciseNames += [exercise]
            print("Which do you want?" )
            print()
            print(results)
        #####PRINT POSSIBLE FOOD ITEMS AND ASK WHICH THEY WANT
        ###THEN, APPEND THEIR ANSWER TO THE DATA
        ###LATER, YOU NEED TO BE ABLE TO EDIT DICTIONARY
            print()
            result = results[0] #CHANGE THIS
            print("Okay, you selected %s" % result[0])#CHANGE THIS
            if self.weight<=130:
                (self.calpermin).append(result[1][0]/minutes)
            elif self.weight <= 155:

                (self.calpermin).append(result[1][1]/minutes)
            elif self.weight <= 180:
                (self.calpermin).append(result[1][2]/minutes)
            else:
                (self.calpermin).append(result[1][3]/minutes)
        #Costs
        (self.happy).append(int(input("How happy does this workout make you?(From 1-10) ")))
        #checks if we want to add another exercise
        if 1 < self.countvariables < self.max-1:
            anotherinput = input("Do you want to add another workout?(Yes or No) ")
            if anotherinput == "No":
                self.anotherinput = False
                minimum_time = roundHalfUp(self.Mcalories[0]/max(self.calpermin))
                statement = "How much time are you willing to spend?(in minutes) Value must be larger than %0.1f minutes. " %
```

```
minimum_time
                (self.Wtime).append(int(input(statement)))
        else:
        #makes sure there is more than one variable
            if self.countvariables > 1:
                self.anotherinput = False
                minimum_time = self.Mcalories[0]/max(self.calpermin)
                statement = "How much time are you willing to spend?(in minutes) Value must be larger than %0.1f minutes. " % minimum_time
                (self.Wtime).append(int(input(statement)))
    def checkDictioanryOfExercise(self, search):
        search = search.lower()
        results = []
        for exercise in EXERCISE.dictionary_of_exercise:
            if search in exercise.lower():
                results.append([exercise,
```

```python
EXERCISE.dictionary_of_exercise[exercise]])
        if results == []:
            return False
        return results

    def newExerciseItem(self, name):
        exercise = name
        self.exerciseNames += [exercise]
        minutes = 60
        calperhour = int(input("How many calories per hour do you burn from this exercise?"))
        (self.calpermin).append(calperhour/minutes)
        #these are burned calories for different weights
        weightCalories = [None,None,None,None]
        if self.weight<=130:
            weightCalories[0] = calperhour
        elif self.weight <= 155:
            weightCalories[1] = calperhour
        elif self.weight <= 180:
            weightCalories[2] =
calperhour
        else: weightCalories[3] = calperhour
        Exercise.addToExerciseDictionary
        #these are the extra calories lost/gained with each calorie bracket
        additional = 40
        #converts all nones to proper calories
        while None in weightCalories:
            for element in range(len(weightCalories)):
                if weightCalories[element] ==None:
                    #edge case first element
                    if element == 0:
                        if weightCalories[element+1] != None:
                            weightCalories[0] = \
                            weightCalories[element+1] - additional
                    #edge case last element
                    elif element == 3:
                        if
```

```
weightCalories[element-1] != None:
                            weightCalories[3] = \
                            weightCalories[element-1] + additional
                    else:
                        if weightCalories[element-1] != None:
                            weightCalories[element] = \
                            weightCalories[element-1] + additional
                        elif weightCalories[element+1] != None:
                            weightCalories[element] = \
                            weightCalories[element+1] - additional
        dictElement = [exercise, tuple(weightCalories)]
        Exercise.addToExerciseDictionary(dictElement)
    @staticmethod
    def addToExerciseDictionary(exercise):
        #finds where to add it in the dictionary
        with open("Databases/Final_Databases/Exer
cise_Database_Final.py") as file:
            data = file.readlines()
        dictionary = data[1]
        index = data[1].index("{")
        #creates entry
        entryName = exercise[0]
        entryTuple = exercise[1]
        dictionary = data[0]+ dictionary[:index+1] + "'%s':%s," % \
        (str(entryName), str(entryTuple))+ dictionary[index+1:]
        #writes newdictionary
        file = open("Databases/Final_Databases/Exercise_Database_Final.py", "w")
        file.write(dictionary)

        file.close()
        #reloads dictionary
        imp.reload(EXERCISE)
    #solves for lambda and then can be used to get x and y values
    def lambdaConstraintSolver(self):
        #these variables are used to provide constraints on linear solver
        epsilon = 10**-15
        delta = 10**15
        #calculated function of legranage optimization (derviation can be shown)
        def function(x):
            def lambdaSolver(x,al):
                alpha = al
                try: return (x/alpha)**(1/(alpha-1))
                #if we get a divide by 0 error, lambda becomes 1
                except: return 1
            equation = 0
            for i in self.alpha:
                equation += lambdaSolver(x,i)
            return equation - self.Wtime[0]
        return brenth(function, epsilon, delta)

    def getValuesFromLambda(self):
        L = self.lambdaConstraintSolver() #lambda
        #these are calculated values for alpha
        exerciseList = []
        for i in self.alpha:
            alpha = i
            exerciseList += [(L/alpha)**(1/(alpha-1))]
        print(self.alpha)
        print(exerciseList)
        better_exercise = max(exerciseList)
        better = exerciseList.index(better_exercise) + 1
        if self.discrete == True:
            return "Exercise %s is the best choice" % better
        else:
            if len(exerciseList) == 2:
                return "You should do %s for %0.2f minutes and %s for %0.2f minutes"\
                % (self.exerciseNames[0],exerciseList[0], self.exerciseNames[1], exerciseList[1])
            elif len(exerciseList) == 3:
                return "You should do %s for %0.2f minutes, %s for %0.2f minutes, and %s for %0.2f minutes" % (self.exerciseNames[0], exerciseList[0], self.exerciseNames[1], exerciseList[1],
self.exerciseNames[2], exerciseList[2])
            elif len(exerciseList) == 4:
                return "You should do %s for %0.2f minutes, %s for %0.2f minutes, %s for %0.2f minutes, and %s for %0.2f minutes"\
                % (self.exerciseNames[0],exerciseList[0], self.exerciseNames[1], exerciseList[1], self.exerciseNames[2], exerciseList[2], self.exerciseNames[3], exerciseList[3])
            elif len(exerciseList) == 5:
                return "You should do %s for %0.2f minutes, %s for %0.2f minutes, %s for %0.2f minutes, %s for %0.2f minutes, and %s for %0.2f minutes"\
                % (self.exerciseNames[0], exerciseList[0], self.exerciseNames[1], exerciseList[1], self.exerciseNames[2], exerciseList[2], self.exerciseNames[3], exerciseList[3], self.exerciseNames[4],
exerciseList[4])

##################################
def runExercise():
    exercise = Exercise()
    exercise.collectInitialself()
    exercise.collectExercise()
    #collect the data until the user says no more
    while exercise.anotherinput and exercise.countvariables<=exercise.max
        exercise.collectExercise()
    exercise.getAlpha()
    print(exercise.getValuesFromLambda())
#################################
```