```
# /Users/Kamyar/Box Sync/Computer Science/Term project/TermProjectFinalMath.py
001| #Name:Kamyar Ghiam
002| #Project: Deciduo
003| #15-112 -- Fall 2017
004|
005| #Food Database
006| import Databases.Final_Databases.FoodDatabase as FOOD
007| #Exercise Database
008| import Databases.Final_Databases.Exercise_Database_Final as EXERCISE
009| #Used for Linear Equations. This is a module from the scipy.optimize package
010| from scipy.optimize import brenth
011| import decimal, string
012| #this is to reload the databases
013| import importlib as imp
014| #taken from course website
015| def roundHalfUp(d):
016|     # Round to nearest with ties going away from zero.
017|     rounding = decimal.ROUND_HALF_UP
018|     return int(decimal.Decimal(d).to_integral_value(rounding=rounding))
```

```
019|
#################################
020|  # Food
021|
#################################
022|
023|  class Food(object):
024|      #module
025|      def __init__(self, discrete
= None, healthy = [], happy = [],\
026|      calories = [], money = [],
Wcalories = [], Wmoney = [],
howhealthy = [],\
027|      howhappy = [], alpha = [],
anotherinput =  True, countvariables
= 0,\
028|      max = 2, min  = 1):
029|          self.max = max #maximum
food items you can ask for
030|          self.min = min
031|          self.discrete =
discrete #checks if the food is
divisible
032|          #Fixed Judgement
033|          self.healthy = healthy
034|          self.happy = happy
035|          #Costs
036|          self.calories =
calories
037|          self.money = money
038|          #Constraints (W =
willing to eat or pay)
039|          self.Wcalories =
```

```
Wcalories
040|            self.Wmoney = Wmoney
041|            #Variables judgement
042|            self.howhealthy =
howhealthy
043|            self.howhappy =
howhappy
044|            #Algorithm
045|            self.alpha = alpha
046|            #Checks if user wants
to input another food
047|            self.anotherinput =
anotherinput
048|            #Counts how many food
items were added
049|            self.countvariables =
countvariables
050|            #collects food items
051|            self.foods = []
052|            #this is for the
machine learning algorithm
053|            self.learn = 1.5
054|        #solves for lambda and then
can be used to get x and y values
055|        def
lambdaSolverMoneyConstraint(self):
056|            epsilon = 10**-8
057|            delta = 10**8
058|            #calculated function of
legranage optimization (derviation
can be shown)
059|            def function(x):
060|                a1 = self.alpha[0]
```

```
#alpha values
061|                a2 = self.alpha[1]
062|                m1 = self.money[0]
#money constraint
063|                m2 = self.money[1]
064|                return
(m1)*(((x*m1)/(a1)))**(1/(a1-1))+(m2
)*(((x*m2)/(a2))**(1/(a2-1))) -
self.Wmoney[0]
065|            return brenth(function,
epsilon, delta)
066|        def
getValuesFromLambdaMoney(self):
067|            L =
self.lambdaSolverMoneyConstraint()
#lambda
068|            #these are calculated
values
069|            a1 =  self.alpha[0]
070|            a2 = self.alpha[1]
071|            m1 = self.money[0]
#money constraint
072|            m2 = self.money[1]
073|            first_food =
((L*m1)/(a1))**(1/(a1-1))
074|            second_food =
((L*m2)/(a2))**(1/(a2-1))
075|            return [first_food,
second_food]
076|        def
lambdaSolverCalorieConstraint(self):
077|            epsilon = 10**-8
078|            delta = 10**8
```

```
079|      #calculated function of
legranage optimization (derviation
can be shown)
080|           def function(x):
081|               a1 = self.alpha[0]
#alpha values
082|               a2 = self.alpha[1]
083|               c1 =
self.calories[0] #calorie constraint
084|               c2 =
self.calories[1]
085|               return
(c1)*(((x*c1)/(a1)))**(1/(a1-1))+(c2
)*(((x*c2)/(a2))**(1/(a2-1))) -
self.Wcalories[0]
086|           return brenth(function,
epsilon, delta)
087|
088|      def
getValuesFromLambdaCalorie(self):
089|           L =
self.lambdaSolverCalorieConstraint()
#lambda
090|           #these are calculated
values
091|           a1 =  self.alpha[0]
092|           a2 = self.alpha[1]
093|           c1 = self.calories[0]
#calorie constraint
094|           c2 = self.calories[1]
095|           first_food =
((L*c1)/(a1))**(1/(a1-1))
096|           second_food =
```

```
((L*c2)/(a2))**(1/(a2-1))
097|            return [first_food,
second_food]
098|        def
collectInitialself(self):
099|            #Constraints (W =
willing to eat or pay)
100|
(self.Wcalories).append(int(input("H
ow many calories are you willing to
eat? ")))
101|
(self.Wmoney).append(int(input("How
much money are you willing to spend
in total? ")))
102|            #Variables judgement
103|
(self.howhealthy).append(\
104|                int(input("How
important is it for you to be
healthy in this meal?(From 1-10)
")))
105|
(self.howhappy).append(int(input("Ho
w important is it for you to feel
satisfied from this meal?(from 1-10)
")))
106|            discrete = input("Is
your food divisble? (i.e. do you
want some of each food?) ")
107|            if discrete == "Yes":
108|                self.discrete =
False
```

```
109|            else: self.discrete =
True
110|
111|        def
checkDictioanryOfFood(self, search):
112|            search = search.lower()
113|            results = []
114|            for food in
FOOD.dictionary_of_food:
115|                if search in
food.lower():
116|
results.append([food,
FOOD.dictionary_of_food[food]])
117|            if results == []:
118|                return False
119|            return results
120|
121|        def collectFood(self):
122|            self.countvariables +=
1
123|            #Fixed Judgement
124|            print()
125|            print("Food item %d:" %
self.countvariables)
126|            print()
127|            food = input("What is
the name of your food? ")
128|            results =
self.checkDictioanryOfFood(food)
129|            if results == False:
130|                print("Sorry, we
don't have that food. Please enter
```

```
it manually (we will store this
entry for next time). ")
131|
self.newFoodItem(food)
132|           else:
133|               print("Which do you
want?" )
134|               print()
135|               print(results)
136|           #####PRINT POSSIBLE
FOOD ITEMS AND ASK WHICH THEY WANT
137|           ###THEN, APPEND THEIR
ANSWER TO THE DATA
138|           ###LATER, YOU NEED TO
BE ABLE TO EDIT DICTIONARY
139|               print()
140|               result = results[0]
#CHANGE THIS
141|
self.foods.append(results[0])#CHANGE
THIS
142|               print("Okay, you
selected %s" % result[0])#CHANGE
THIS
143|
(self.healthy).append(result[1][3])
144|
(self.calories).append(result[1][0])
145|
146|           print()
147|           #variable judgemnet
148|
(self.happy).append(int(input(\
```

```
149|          "How happy would it
make you to eat one serving?(From
1-10) ")))
150|
(self.money).append(int(input("How
much does it cost per serving (In
dollars)? ")))
151|          if 1 <
self.countvariables < self.max-1:
152|              anotherinput =
input("Do you want to add another
food item?(Yes or No) ")
153|              if anotherinput ==
"No":
154|
self.anotherinput = False
155|
156|      def newFoodItem(self,food):
157|          name = food
158|          healthy =
int(input("How healthy is this food
item?(From 1-10) "))
159|
(self.healthy).append(healthy)
160|          #Costs
161|          calories =
int(input("How many calories is one
serving? "))
162|
(self.calories).append(calories)
163|          servings = input("How
much is one serving? (for example,
write 1 oz) ")
```

```
164|            serving_number = 0
165|            serving_word = ""
166|            for letter in
range(len(servings)):
167|                if servings[letter]
in string.ascii_letters:
168|                    serving_number
= int(servings[0:letter-1])
169|                    serving_word =
servings[letter:]
170|                    break
171|            food_entry = [name,
(calories, serving_word,
serving_number,healthy)]
172|
self.foods.append(food_entry)
173|
Food.addToFoodDictionary(food_entry)
174|            print(self.foods)
175|
176|        @staticmethod
177|        def
addToFoodDictionary(food):
178|            #finds where to add it
in the dictionary
179|            with
open("Databases/Final_Databases/Food
Database.py") as file:
180|                data =
file.readlines()
181|            dictionary = data[1]
182|            index =
data[1].index("{")
```

```
183|            #creates entry
184|            entryName = food[0]
185|            entryTuple = food[1]
186|            dictionary = data[0]+
dictionary[:index+1] + "'%s':%s," %
\
187|            (str(entryName),
str(entryTuple))+
dictionary[index+1:]
188|            #writes newdictionary
189|            file =
open("Databases/Final_Databases/Food
Database.py", "w")
190|            file.write(dictionary)
191|
192|            file.close()
193|            #reloads dictionary
194|            imp.reload(FOOD)
195|
196|        def getAlpha(self):
197|            h1 = self.howhealthy[0]
198|            h2 = self.howhappy[0]
199|            t1x = self.healthy[0]
200|            t1y = self.healthy[1]
201|            t2x = self.happy[0]
202|            t2y = self.happy[1]
203|            alpha1 =
1/(self.learn*(h1*t1y + h2*t2y))
#This is a calculated values based
on a formula I created
204|            alpha2 = 1/(h1*t1x +
h2*t2x)
205|
```

```
(self.alpha).append(alpha1)
206|
(self.alpha).append(alpha2)
207|     def
compareUtilityFunction(self):
208|         #finds the smaller
utility inside constraints
209|         def utilityFunction(x,
y):
210|             return
x**(self.alpha[0]) +
y**(self.alpha[1])
211|         x1 =
self.money_solution[0]
212|         y1 =
self.money_solution[1]
213|         x2 =
self.calorie_solution[0]
214|         y2 =
self.calorie_solution[1]
215|         if utilityFunction(x1,
y1) <= utilityFunction(x2, y2):
216|             answer = [x1,y1]
217|         else: answer = [x2,y2]
218|         #finds the better food
219|         if answer[0] >=
answer[1]:
220|             better_name =
self.foods[0][0]
221|         else:
222|             better_name =
self.foods[1][0]
223|         print(answer)
```

```
224|            if self.discrete ==
True:
225|                return "Eating %s
is the better choice" % better_name
226|            else:
227|                servings1 =
answer[0]*self.foods[0][1][2]
228|                servings2 =
answer[1]*self.foods[1][1][2]
229|                food1 =
self.foods[0][0]
230|                food2 =
self.foods[1][0]
231|                servingword1 =
self.foods[0][1][1]
232|                servingword2 =
self.foods[1][1][1]
233|                return "For %s, I
would have %0.2f (serving %s), and
for %s, I would have %0.2f (serving
%s)"\
234|            % (food1,
servings1,servingword1, food2,
servings2, servingword2)
235|                """
236|        #machine learning feature
237|        def adjustAlgorithm(self):
238|                """
239|    ################
240|
241|    def runFood():
242|        food = Food()
243|        food.collectInitialself()
```

```
244|      food.collectFood()
245|      while food.anotherinput and
food.countvariables < food.max:
246|          food.collectFood()
247|      food.getAlpha()
248|      food.money_solution =
food.getValuesFromLambdaMoney()
249|      food.calorie_solution =
food.getValuesFromLambdaCalorie()
250|
print(food.compareUtilityFunction())
251|
252|  ############
253|
254|
255|
256|
257|
258|
259|
260|
##################################
261|  # Exercise
262|
##################################
263|
264|  #could not use interticance
because the optimization algorithm
is different
265|  class Exercise(object):
266|      def __init__(self, max = 5,
min = 1, discrete = None, happy =
[],\
```

```
267|      calpermin = [], Wtime = [],
Mcalories = [], howhappy = [], alpha
= [],\
268|      anotherinput =  True,
countvariables = 0, age = 0, weight
= 0):
269|          #module
270|          self.max = max #maximum
exercise items you can ask for
271|          self.min =  min
272|          self.discrete =
discrete #checks if you can do
multiple exercises
273|          #Fixed Judgement
274|          self.happy = happy
275|          self.calpermin =
calpermin
276|          #Constraints (W =
willing time. M is minimum calories
wanted to be burned)
277|          self.Wtime = Wtime
278|          self.Mcalories =
Mcalories
279|          #Variables judgement
280|          self.howhappy =
howhappy
281|          #Algorithm
282|          self.alpha = alpha
283|          self.anotherinput =
anotherinput #Checks if user wants
to input another exercise
284|          self.countvariables =
countvariables #Counts how many
```

```
exercise items were added
285|            self.age = age
286|            self.weight = weight
287|            self.exerciseNames = []
288|
289|        def getAlpha(self):
290|            maximum = 11 #I chose
11 because I don't want to get a
divide by 0 error when subtracting
maximum (10)
291|            h = (maximum -
self.howhappy[0])
292|            for i in
range(len(self.happy)):
293|                h1 = maximum -
self.happy[i]
294|                alpha = 1/(2*h*h1)
#calculated values based on a
formula I created
295|
(self.alpha).append(alpha)
296|
297|
298|        def
collectInitialself(self):
299|            #Constraints (W =
willing to eat or pay)
300|            (self.weight) =
(int(input("How much do you weigh?
")))
301|            (self.age) =
(int(input("How old are you? ")))
302|
```

```
(self.Mcalories).append(int(input("H
ow many calories do you want to
burn? ")))
303|            #Variables judgement
304|
(self.howhappy).append(\
305|                int(input("How
important is it for you to enjoy
this workout?(From 1-10) ")))
306|                discrete = input("Do
you want to do multiple workouts?
(i.e. split your time between
workouts?) ")
307|                if discrete == "Yes":
308|                    self.discrete =
False
309|                else: self.discrete =
True
310|
311|        def collectExercise(self):
312|            minutes = 60
313|            self.countvariables +=
1
314|            #Fixed Judgement
315|            print()
316|            print("Exercise item
%d:" % self.countvariables)
317|            print()
318|            exercise = input("What
is the name of your exercise? ")
319|            results =
self.checkDictioanryOfExercise(exerc
ise)
```

```
320|            if results == False:
321|                print("Sorry, we
don't have that exercise. Please
enter it manually.")
322|
self.newExerciseItem(exercise)
323|            else:
324|                self.exerciseNames
+= [exercise]
325|                print("Which do you
want?" )
326|                print()
327|                print(results)
328|            #####PRINT POSSIBLE
FOOD ITEMS AND ASK WHICH THEY WANT
329|            ###THEN, APPEND THEIR
ANSWER TO THE DATA
330|            ###LATER, YOU NEED TO
BE ABLE TO EDIT DICTIONARY
331|                print()
332|                result = results[0]
#CHANGE THIS
333|                print("Okay, you
selected %s" % result[0])#CHANGE
THIS
334|                if
self.weight<=130:
335|
(self.calpermin).append(result[1]
[0]/minutes)
336|                elif self.weight <=
155:
337|
```

```
(self.calpermin).append(result[1]
[1]/minutes)
338|                elif self.weight <=
180:
339|
(self.calpermin).append(result[1]
[2]/minutes)
340|                else:
(self.calpermin).append(result[1]
[3]/minutes)
341|            #Costs
342|
(self.happy).append(int(input("How
happy does this workout make you?
(From 1-10) ")))
343|            #checks if we want to
add another exercise
344|            if 1 <
self.countvariables < self.max-1:
345|                anotherinput =
input("Do you want to add another
workout?(Yes or No) ")
346|                if anotherinput ==
"No":
347|
self.anotherinput = False
348|                    minimum_time =
roundHalfUp(self.Mcalories[0]/max(se
lf.calpermin))
349|                    statement =
"How much time are you willing to
spend?(in minutes) Value must be
larger than %0.1f minutes. " %
```

```
minimum_time
350|
(self.Wtime).append(int(input(statem
ent)))
351|           else:
352|           #makes sure there is
more than one variable
353|                   if
self.countvariables > 1:
354|
self.anotherinput = False
355|                    minimum_time =
self.Mcalories[0]/max(self.calpermin
)
356|                    statement =
"How much time are you willing to
spend?(in minutes) Value must be
larger than %0.1f minutes. " %
minimum_time
357|
(self.Wtime).append(int(input(statem
ent)))
358|       def
checkDictioanryOfExercise(self,
search):
359|           search = search.lower()
360|           results = []
361|           for exercise in
EXERCISE.dictionary_of_exercise:
362|                   if search in
exercise.lower():
363|
results.append([exercise,
```

```
EXERCISE.dictionary_of_exercise[exer
cise]])
364|            if results == []:
365|                return False
366|            return results
367|
368|        def newExerciseItem(self,
name):
369|            exercise = name
370|            self.exerciseNames +=
[exercise]
371|            minutes = 60
372|            calperhour =
int(input("How many calories per
hour do you burn from this exercise?
"))
373|
(self.calpermin).append(calperhour/m
inutes)
374|            #these are burned
calories for different weights
375|            weightCalories =
[None,None,None,None]
376|            if self.weight<=130:
377|                weightCalories[0] =
calperhour
378|            elif self.weight <=
155:
379|                weightCalories[1] =
calperhour
380|            elif self.weight <=
180:
381|                weightCalories[2] =
```

```
calperhour
382|          else: weightCalories[3]
= calperhour
383|
Exercise.addToExerciseDictionary
384|          #these are the extra
calories lost/gained with each
calorie bracket
385|          additional = 40
386|          #converts all nones to
proper calories
387|          while None in
weightCalories:
388|                for element in
range(len(weightCalories)):
389|                    if
weightCalories[element] ==None:
390|                        #edge case
first element
391|                        if element
== 0:
392|                            if
weightCalories[element+1] != None:
393|
weightCalories[0] = \
394|
weightCalories[element+1] -
additional
395|                        #edge case
last element
396|                        elif
element == 3:
397|                            if
```

```
weightCalories[element-1] != None:
398|
weightCalories[3] = \
399|
weightCalories[element-1] +
additional
400|                         else:
401|                             if
weightCalories[element-1] != None:
402|
weightCalories[element] = \
403|
weightCalories[element-1] +
additional
404|                             elif
weightCalories[element+1] != None:
405|
weightCalories[element] = \
406|
weightCalories[element+1] -
additional
407|             dictElement =
[exercise, tuple(weightCalories)]
408|
Exercise.addToExerciseDictionary(dic
tElement)
409|         @staticmethod
410|         def
addToExerciseDictionary(exercise):
411|             #finds where to add it
in the dictionary
412|             with
open("Databases/Final_Databases/Exer
```

```
cise_Database_Final.py") as file:
413|                data =
file.readlines()
414|            dictionary = data[1]
415|            index =
data[1].index("{")
416|            #creates entry
417|            entryName = exercise[0]
418|            entryTuple =
exercise[1]
419|            dictionary = data[0]+
dictionary[:index+1] + "'%s':%s," %
\
420|                (str(entryName),
str(entryTuple))+
dictionary[index+1:]
421|            #writes newdictionary
422|            file =
open("Databases/Final_Databases/Exer
cise_Database_Final.py", "w")
423|            file.write(dictionary)
424|
425|            file.close()
426|            #reloads dictionary
427|            imp.reload(EXERCISE)
428|        #solves for lambda and then
can be used to get x and y values
429|        def
lambdaConstraintSolver(self):
430|            #these variables are
used to provide constraints on
linear solver
431|            epsilon = 10**-15
```

```
432|            delta = 10**15
433|            #calculated function of
legranage optimization (derviation
can be shown)
434|            def function(x):
435|                def
lambdaSolver(x,al):
436|                    alpha = al
437|                    try: return
(x/alpha)**(1/(alpha-1))
438|                    #if we get a
divide by 0 error, lambda becomes 1
439|                    except: return
1
440|                equation = 0
441|                for i in
self.alpha:
442|                    equation +=
lambdaSolver(x,i)
443|                return equation -
self.Wtime[0]
444|            return brenth(function,
epsilon, delta)
445|
446|        def
getValuesFromLambda(self):
447|            L =
self.lambdaConstraintSolver()
#lambda
448|            #these are calculated
values for alpha
449|            exerciseList = []
450|            for i in self.alpha:
```

```
451|                 alpha = i
452|                 exerciseList +=
[(L/alpha)**(1/(alpha-1))]
453|             print(self.alpha)
454|             print(exerciseList)
455|             better_exercise =
max(exerciseList)
456|             better =
exerciseList.index(better_exercise)
+ 1
457|             if self.discrete ==
True:
458|                 return "Exercise %s
is the best choice" % better
459|             else:
460|                 if
len(exerciseList) == 2:
461|                     return "You
should do %s for %0.2f minutes and
%s for %0.2f minutes"\
462|                     %
(self.exerciseNames[0],exerciseList[
0], self.exerciseNames[1],
exerciseList[1])
463|                 elif
len(exerciseList) == 3:
464|                     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])
465|                elif
len(exerciseList) == 4:
466|                    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"\
467|                    %
(self.exerciseNames[0],exerciseList[
0], self.exerciseNames[1],
exerciseList[1],
self.exerciseNames[2],
exerciseList[2],
self.exerciseNames[3],
exerciseList[3])
468|                elif
len(exerciseList) == 5:
469|                    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"\
470|                    %
(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])
471|
472|
##################################
473|  def runExercise():
474|      exercise = Exercise()
475|
exercise.collectInitialself()
476|      exercise.collectExercise()
477|      #collect the data until the
user says no more
478|      while exercise.anotherinput
and
exercise.countvariables<=exercise.max
479|
exercise.collectExercise()
480|      exercise.getAlpha()
481|
print(exercise.getValuesFromLambda()
)
482|
##################################
```