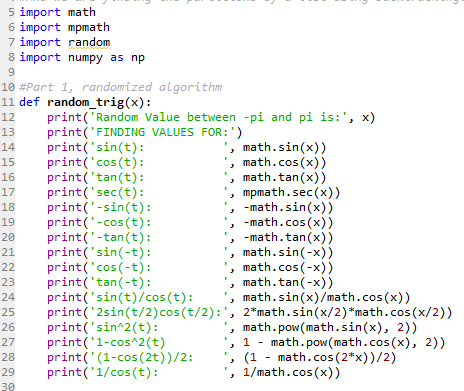
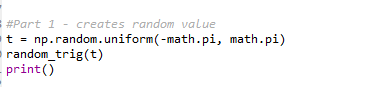
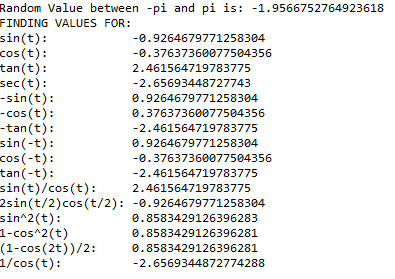
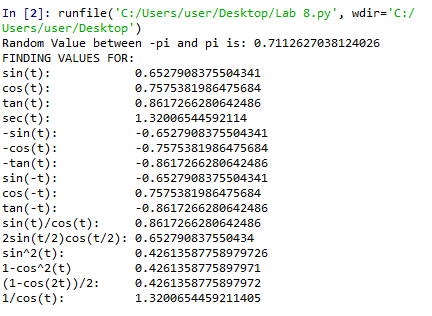
Lab 8

For this lab, we needed to create two different methods from scratch. One is based on solving several trigonometric problems in given a random value between (-pi, pi). Each function holds the same random value, so if I randomly generated 2, all the trig. Functions would input 2 as a value. For problem two, we need to create a backtracking algorithm that receives one set of integers and from that we need to create discover if that set can be split up into two different, but equal subsets. The two subsets cannot share any integers, as such that if 2 is in the base subset, it will either go into Subset-1, or Subset-2, but not both.

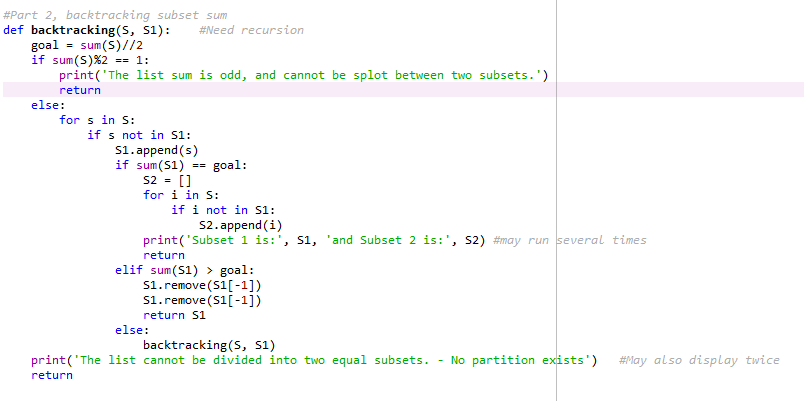
For problem 1, first we need to create a method the takes a variable and solves 16 different trigonometric problems with it. In order for Python to solve the trigonometric problems, we need to import math and mpmath (for sec). With this, we just write out the format for the problems and we only need the variable. To get the random value between -pi and pi, we use the function np.random.uniform(-math.pi,math.pi). We then use this variable created using this to call on the method that solves the trigonometric identities.

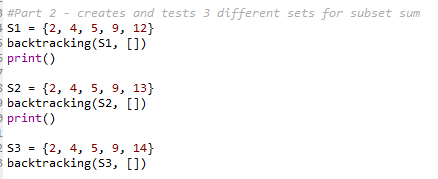


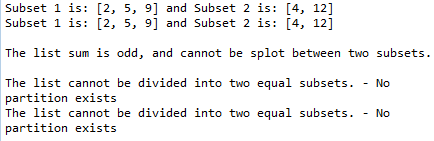




For problem 2, we first need to check if the set can be separated into two equal subsets. To do this, we find the sum of the set, and if its even, we can then go on the check if we can divide it up accordingly. For our backtracking method, we need to call on two variables, one of them being the set of integers, and the other bring an empty list designed to be subset-1. Within our function, we create a goal which is equal to half the sum of this list, as we want our subsets to equal that. At the beginning of the function, we check if the sum of the list is even. If it’s not, we end the method and report that the sum of the set is odd, and thus cannot be divided by two. Otherwise, we continue on, and use a for statement to traverse the list. We check value is in subset-1, and if its not, we append it to the subset. We then check if the sum of subset-1 is equal to the goal. If it is, we append all values in subset-1 into a new list subset-2, and report that we found the two equal subsets. Going back, if subset-1 is not equal to the goal, we check if its sum is greater than it. If it is, we remove the last two values in the list and return. We have to remove two, as only removing one will result in the same problem of having a bigger value than the goal. Lastly, if the sum is not bigger, or equal to the goal, we recursively call on the backtracking method. And if no combination of values is found to be equal to the goal, we report that no partition exists, and we return the method.







In conclusion, I learned that the math imports are extremely convenient, and how to create a backtracking algorithm. I learned how to solve a subset-sum problem and divide up lists of values.

Appendix:

#Patrick Brannan

#Last Edited on 5/11/2019 - Due 5/9/2019

#For this program, we are implementing a trigonometric functions with random values

#And we are finding the partitions of a list using backtracking.

import math

import mpmath

import random

import numpy as np

#Part 1, randomized algorithm

def random\_trig(x):

print('Random Value between -pi and pi is:', x)

print('FINDING VALUES FOR:')

print('sin(t): ', math.sin(x))

print('cos(t): ', math.cos(x))

print('tan(t): ', math.tan(x))

print('sec(t): ', mpmath.sec(x))

print('-sin(t): ', -math.sin(x))

print('-cos(t): ', -math.cos(x))

print('-tan(t): ', -math.tan(x))

print('sin(-t): ', math.sin(-x))

print('cos(-t): ', math.cos(-x))

print('tan(-t): ', math.tan(-x))

print('sin(t)/cos(t): ', math.sin(x)/math.cos(x))

print('2sin(t/2)cos(t/2):', 2\*math.sin(x/2)\*math.cos(x/2))

print('sin^2(t): ', math.pow(math.sin(x), 2))

print('1-cos^2(t) ', 1 - math.pow(math.cos(x), 2))

print('(1-cos(2t))/2: ', (1 - math.cos(2\*x))/2)

print('1/cos(t): ', 1/math.cos(x))

#Part 2, backtracking subset sum

def backtracking(S, S1): #Need recursion

goal = sum(S)//2

if sum(S)%2 == 1:

print('The list sum is odd, and cannot be splot between two subsets.')

return

else:

for s in S:

if s not in S1:

S1.append(s)

if sum(S1) == goal:

S2 = []

for i in S:

if i not in S1:

S2.append(i)

print('Subset 1 is:', S1, 'and Subset 2 is:', S2) #may run several times

return

elif sum(S1) > goal:

S1.remove(S1[-1])

S1.remove(S1[-1])

return S1

else:

backtracking(S, S1)

print('The list cannot be divided into two equal subsets. - No partition exists') #May also display twice

return

#Part 1 - creates random value

t = np.random.uniform(-math.pi, math.pi)

random\_trig(t)

print()

#Part 2 - creates and tests 3 different sets for subset sum

S1 = {2, 4, 5, 9, 12}

backtracking(S1, [])

print()

S2 = {2, 4, 5, 9, 13}

backtracking(S2, [])

print()

S3 = {2, 4, 5, 9, 14}

backtracking(S3, [])