**Lab Objective**

For this lab, I need to make a program that uses randomized algorithms that will test all the trigonometric expressions and see if they are equal to each other. After that I need to use backtracking to see if a partition exists between sets.

**Proposed Solution**

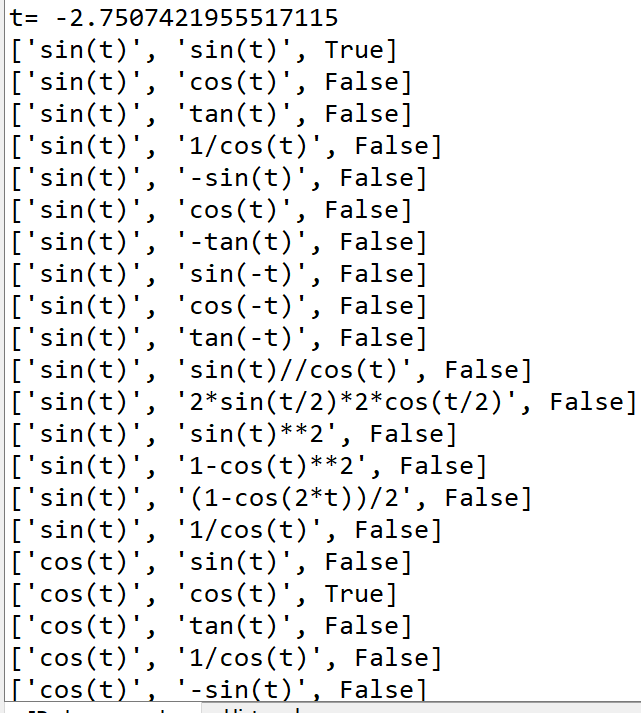
For this first part of the program, I need to be able to check all the trig functions to each other and see if they are equal. To do this, I need to modify some of the class code given in order to find the equalities of all the functions. Since I’m going through all the functions and comparing them to each other then this will require the use of a for loop to iterate through the functions. Since these functions are strings I need to convert them integers and see them as math equations.

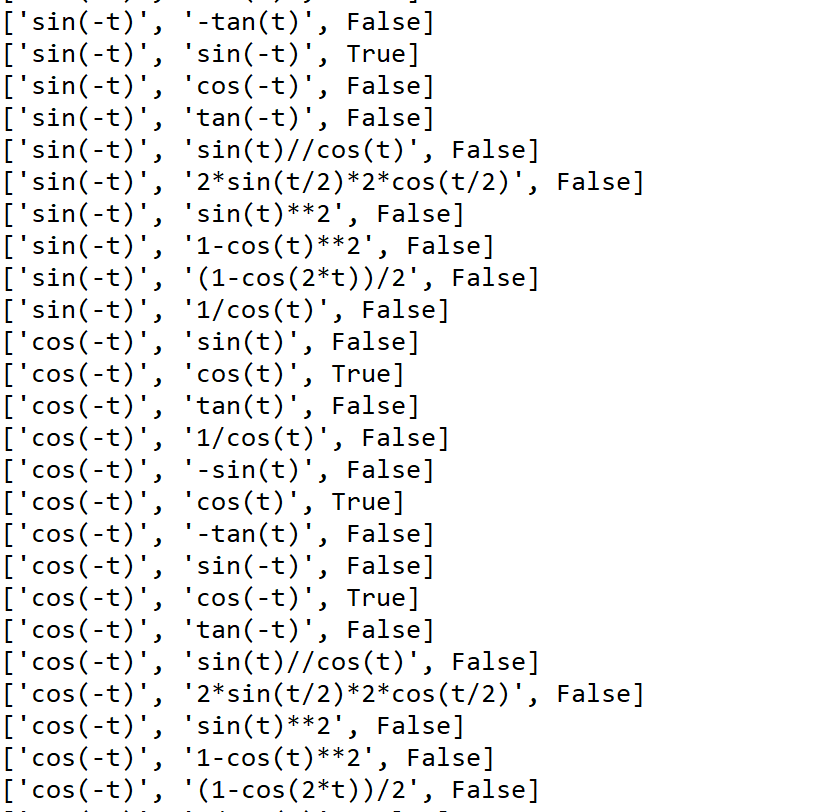
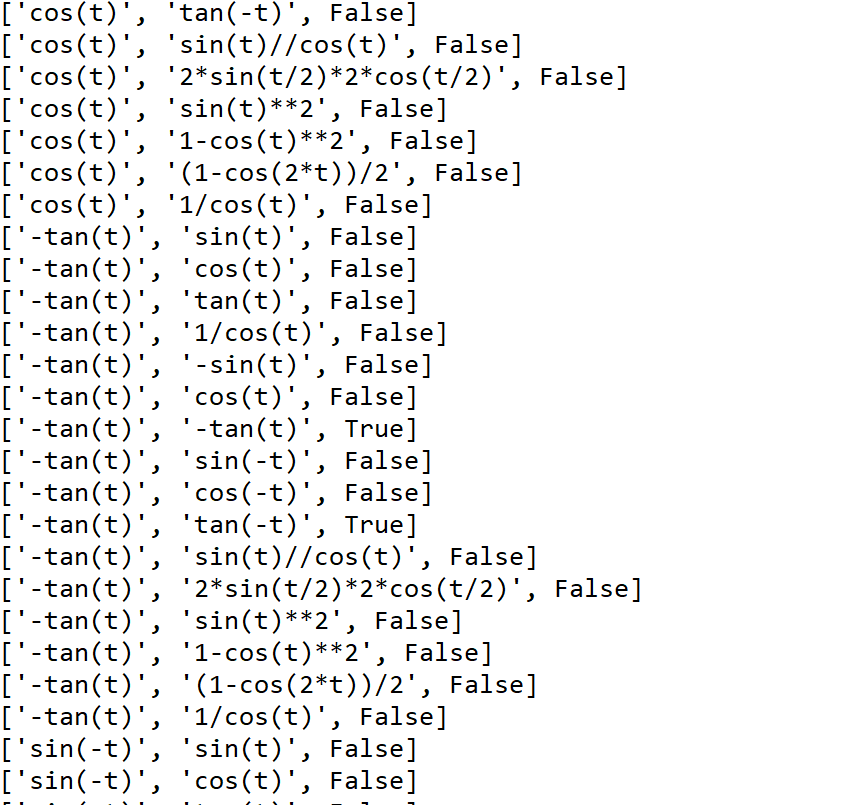
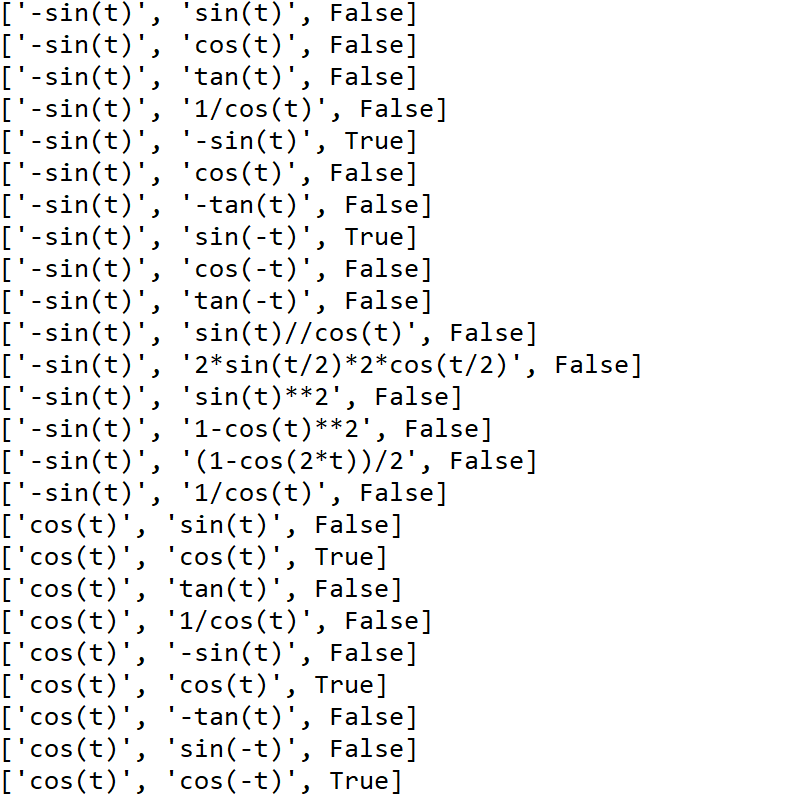
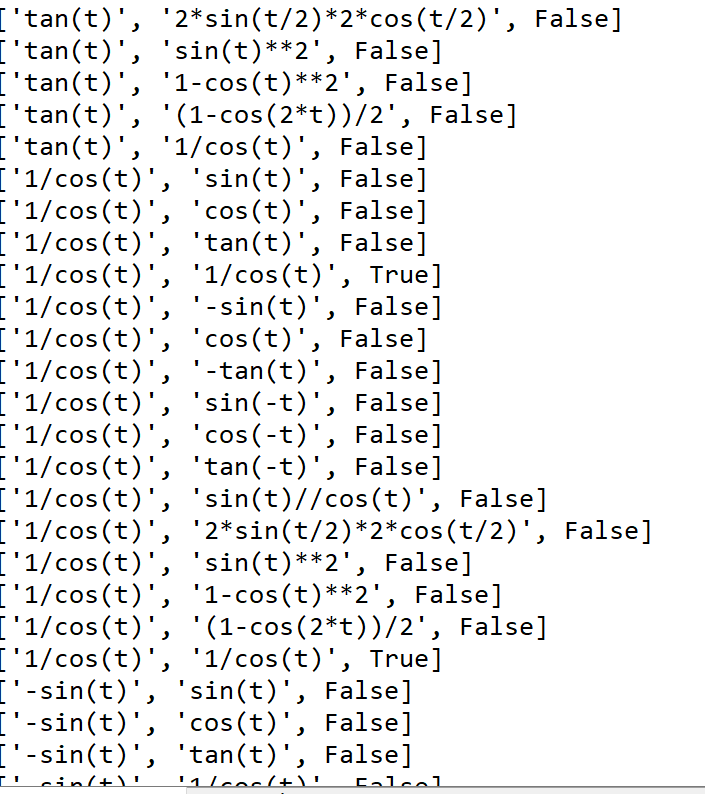
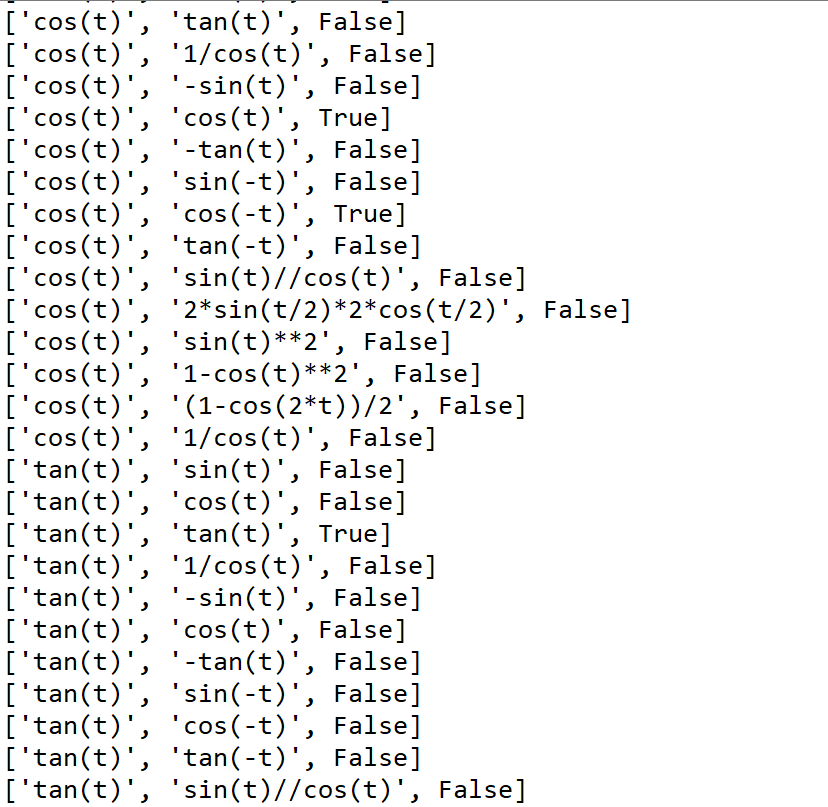
**Implementation**

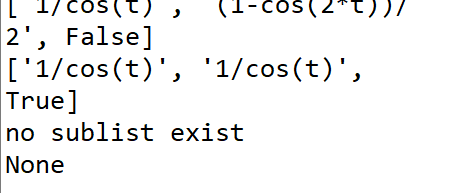
To begin my Implementation, I used the equal method that was given and modified it. I first made a variable called res to store an empty list. After that I used variable t to store the math pi functions and then I print them. After that I need to traverse the list of functions that was passed into the method with a double for loop and in the for loops I made two variables that store the eval of the index at a list. After storing those values from the list, I then need to compare them and see if they’re equal. If they are equal, then those values get appended to the res and the Boolean datatype true gets appended too. However, if they are not equal then the values still get appended to the list, but with the Boolean datatype false.

**Tracing**

To trace this, I made a list of the entire functions of trigonometry and passed it in to my method that checks if they are equal. In the method the list that will store the results of the method will begin as empty. Then the method will traverse the list of functions with a double for loop and store the values from the indices of the list. After obtaining the values the if statement checks if they two values at the index are the same and if they are then they are both appended to the list with the value true and if not then the values of the list are appended to the list but with the value false as they are not the same.







**Academic dishonesty**

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion. Cheating may involve copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data (for example program outputs) in laboratory reports. Plagiarism occurs when someone represents the work or ideas of another person as his/her own. Collusion involves collaborating with another person to commit an academically dishonest act. Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

**Appendix**

# -\*- coding: utf-8 -\*-

"""

Created on Wed May 8 21:42:32 2019

@author: Alexis Navarro

Olac Fuentes

CS 2302

Purpose: make a code to find if the equations are equal to each other and to use

backtracking to find the partition of the subset if possible.

"""

import random

import numpy as np

from math import \*

import math

import time

#METHOD TO FIND IF THE EQUATIONS ARE EQUAL TO EACH OTHER

def equal(L):

res=[] #LIST TO HOLD THE RESULTS

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

print('t=',t)

for i in range(len(L)):

for y in range(len(L)):

y1 = eval(L[i])

y2 = eval(L[y])

if y1 == y2: # IF METHOD TO CHECK IF THE EQUATIONS ARE EQUAL

res.append([L[i],L[y],True])

else:

res.append([L[i],L[y],False])

return res

'''

#method to find the subsetSum of the equations to be used to see if they are equal.

def subsetSum(L,i,sumL):

if sum ==0:

return True

if i == 0 and sum !=0:

return False

if L[i-1]>sum:

return subsetSum(L,i-1,sum)

return subsetSum(L,i-1,sum) or subsetSum(L,i-1,sum-L[i-1])

'''

# METHOD TO BACKTRACK

def subsetsumP(S,last,goal):

if goal ==0:

return []

if goal<0 or last<0:

return []

subset =[] # Take S[last]

if S[last]>goal:

return subsetsumP(S,last-1,goal)

else:

subset.append(S[last])

return subsetsumP (S,last-1,goal) # Don't take S[last]

#METHOD TO FIND THE PARTITION OF THE SUBSETS

def partition(S,index):

sumL=0

for i in range(len(S)):

sumL+=S[i]

if sumL % 2 !=0: #IF THE SUM OF THE PASSED LIST IS NOT EVEN THEN THE TWO SUBSETS DON'T CAN'T BE EQUAL IN VALUE

return False

newList = subsetsumP(S,index,sumL//2)# the new list will hold the values that come from the subset sum

sum\_newL=0

sumRemL=0 #HOLDS THE SUM OF THE SUBLIST THAT IS NOT RETURNED

rem=[]

for j in range(len(S)): #INSERTS THE LIST WITH THE VALUES remaining from the new list

if S[j] not in newList:

rem.append(S[j])

for k in range(len(newList)): # traverses the for loop to get the sum of the new list

sum\_newL += newList[k]

for i2 in range(len(rem)): # gets the sum of the remaining list

sumRemL += rem[i2]

if sum\_newL == sumL//2 and sumRemL==sumL//2:

print(newList,rem)

else:

print('no sublist exist')

#------------------------------------------------------------------------------

#COMPARE THE EQUATIONS

F=['sin(t)','cos(t)','tan(t)','1/cos(t)','-sin(t)','cos(t)','-tan(t)','sin(-t)','cos(-t)','tan(-t)','sin(t)//cos(t)','2\*sin(t/2)\*2\*cos(t/2)','sin(t)\*\*2','1-cos(t)\*\*2','(1-cos(2\*t))/2','1/cos(t)']

results = equal(F)

for i in results:

print(i)

#TEST THE PARTITIONS

#S = [2,4,5,9]

S=[2,4,5,9,12]

print(partition(S,len(S)-1))