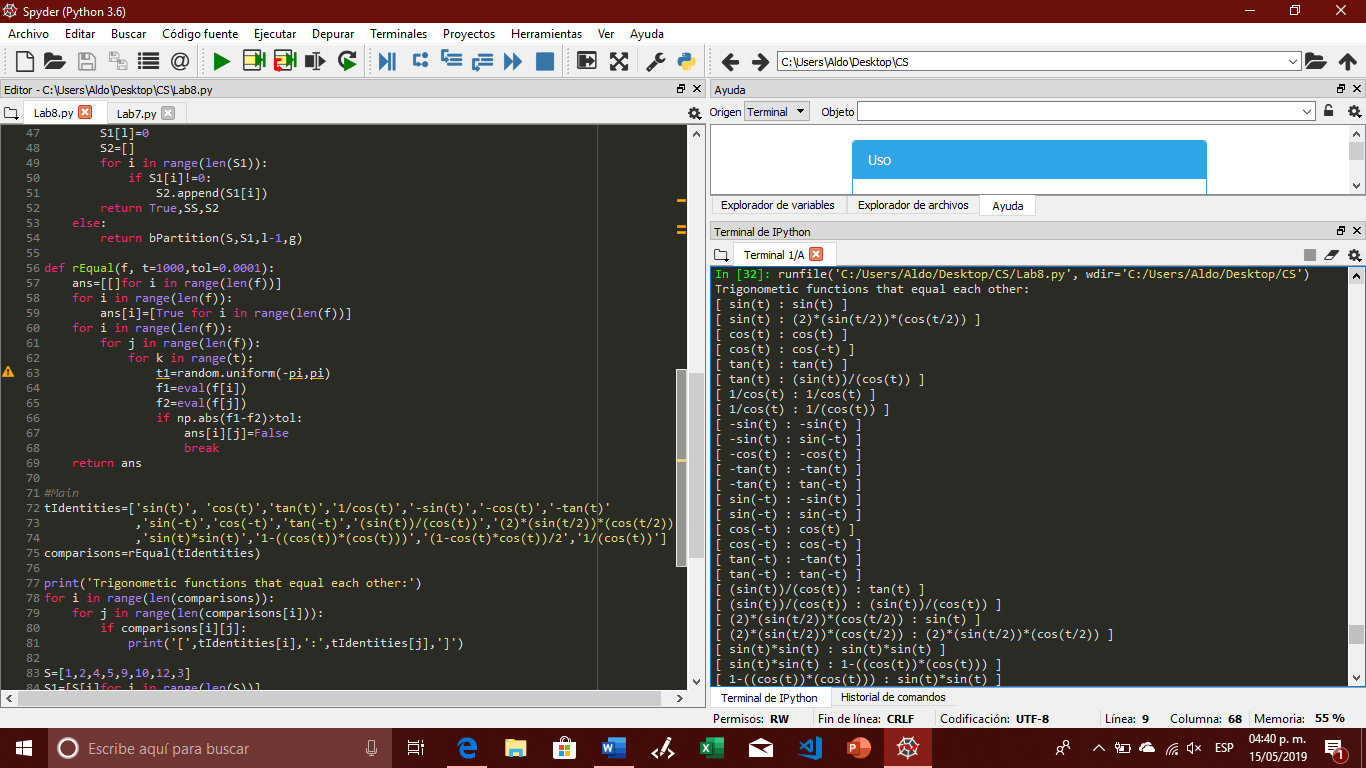
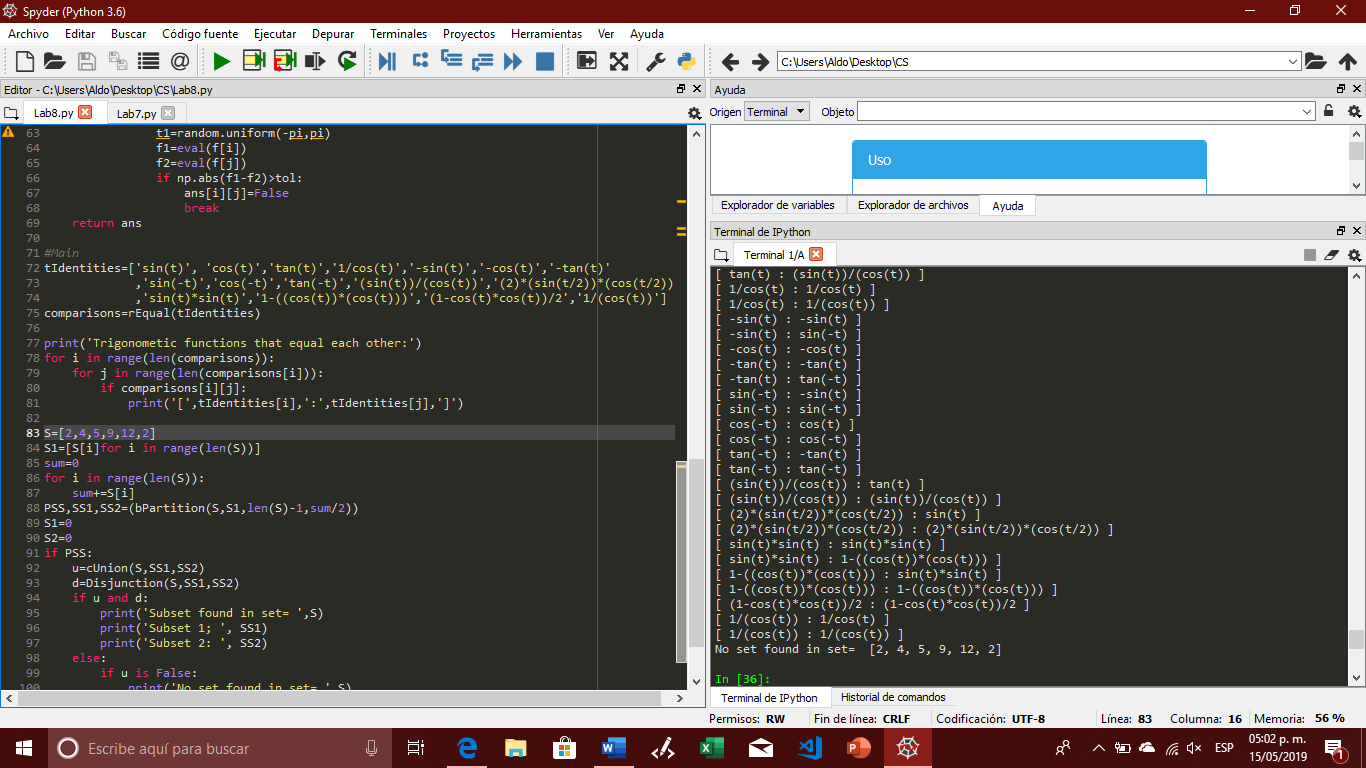
For this lab, we were required to create a program to identify trigonometric identities testing all the combinations in a set of functions from -pi to pi. This process is supposed to be done with a randomized algorithm, using random numbers within that range to test the functions. For the second part of the lab, we need to solve the partition problem. This is, if we have a set of integers S, see if it is possible to divide it into two sets S1 and S2, where the summation of the set S1, is equal to the summation of S2. This problem is supposed to be solved using backtracking.

**Randomization problem:** To solve this problem, I first need to create my set with trigonometrical equations to test. After that, a number is randomly generated form the given range. Then, that number is tested in all the trigonometrical functions, and the outputs are compared to determine which functions have the same outputs.



**Backtracking problem:**  To solve this problem, first we need n initial set, then we divide it into two subsets. But, there are some condition that these sets need to follow, all the elements in the original set S, need to be present in one of the subset, but every element must appear just once in one of the subsets. So, after the subsets are generated, we check to see if they follow the rules. If they do not, we go back to repeat the process until we find a pair of subsets that meet the conditions. Once we find them, we need to check if the summations are the same, if they are not the process is repeated until the subsets meet all the condition are found, or until the limit of tries is reached.



After completing this lab, I feel that I have a better understanding of what backtracking and randomization are, how to use them, and when is a good to use them or not.

import random

import numpy as np

from math import \*

def cUnion(S,S1,S2):

if len(S1)+len(S2)!=len(S):

return False

sum=0

sbSum=0

for i in S1:

if i not in S:

return False

for i in S2:

if i not in S:

return False

return True

def Disjunction(S,S1,S2):

if len(S1)<len(S2):

for i in S1:

if i in S2:

return False

for i in S2:

if i in S1:

return False

return True

def bPartition(S,S1,l,g):

if g==0:

return True,[],[]

if g<0 or l<0:

return False,[],[]

a, SS, S2= bPartition(S,S1,l,g-S[l])

if a:

SS.append(S[l])

S1[l]=0

S2=[]

for i in range(len(S1)):

if S1[i]!=0:

S2.append(S1[i])

return True,SS,S2

else:

return bPartition(S,S1,l-1,g)

def rEqual(f, t=1000,tol=0.0001):

ans=[[]for i in range(len(f))]

for i in range(len(f)):

ans[i]=[True for i in range(len(f))]

for i in range(len(f)):

for j in range(len(f)):

for k in range(t):

t1=random.uniform(-pi,pi)

f1=eval(f[i])

f2=eval(f[j])

if np.abs(f1-f2)>tol:

ans[i][j]=False

break

return ans

#Main

tIdentities=['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))\*(cos(t/2))'

,'sin(t)\*sin(t)','1-((cos(t))\*(cos(t)))','(1-cos(t)\*cos(t))/2','1/(cos(t))']

comparisons=rEqual(tIdentities)

print('Trigonometic functions that equal each other:')

for i in range(len(comparisons)):

for j in range(len(comparisons[i])):

if comparisons[i][j]:

print('[',tIdentities[i],':',tIdentities[j],']')

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

S1=[S[i]for i in range(len(S))]

sum=0

for i in range(len(S)):

sum+=S[i]

PSS,SS1,SS2=(bPartition(S,S1,len(S)-1,sum/2))

S1=0

S2=0

if PSS:

u=cUnion(S,SS1,SS2)

d=Disjunction(S,SS1,SS2)

if u and d:

print('Subset found in set= ',S)

print('Subset 1; ', SS1)

print('Subset 2: ', SS2)

else:

if u is False:

print('No set found in set= ',S)

else:

print('No set found in set= ',S)

else:

print('No set was foundt in set= ',S)

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

Aldo A. Venzor