**CS2302 - Data Structures**

**Spring 2019**

**Lab Report 8**

May 13, 2019

Sebastian Gomez

**Introduction:**

Using random values within the provided limit, compute trigonometric operations to find which ones are identities. Then use the backtracking algorithm to know if there is a possible partition of set of numbers, more especifically, a set of integer. If possible, print the subsets, if not, display to the user that a partition does not exist.

**Implementation:**

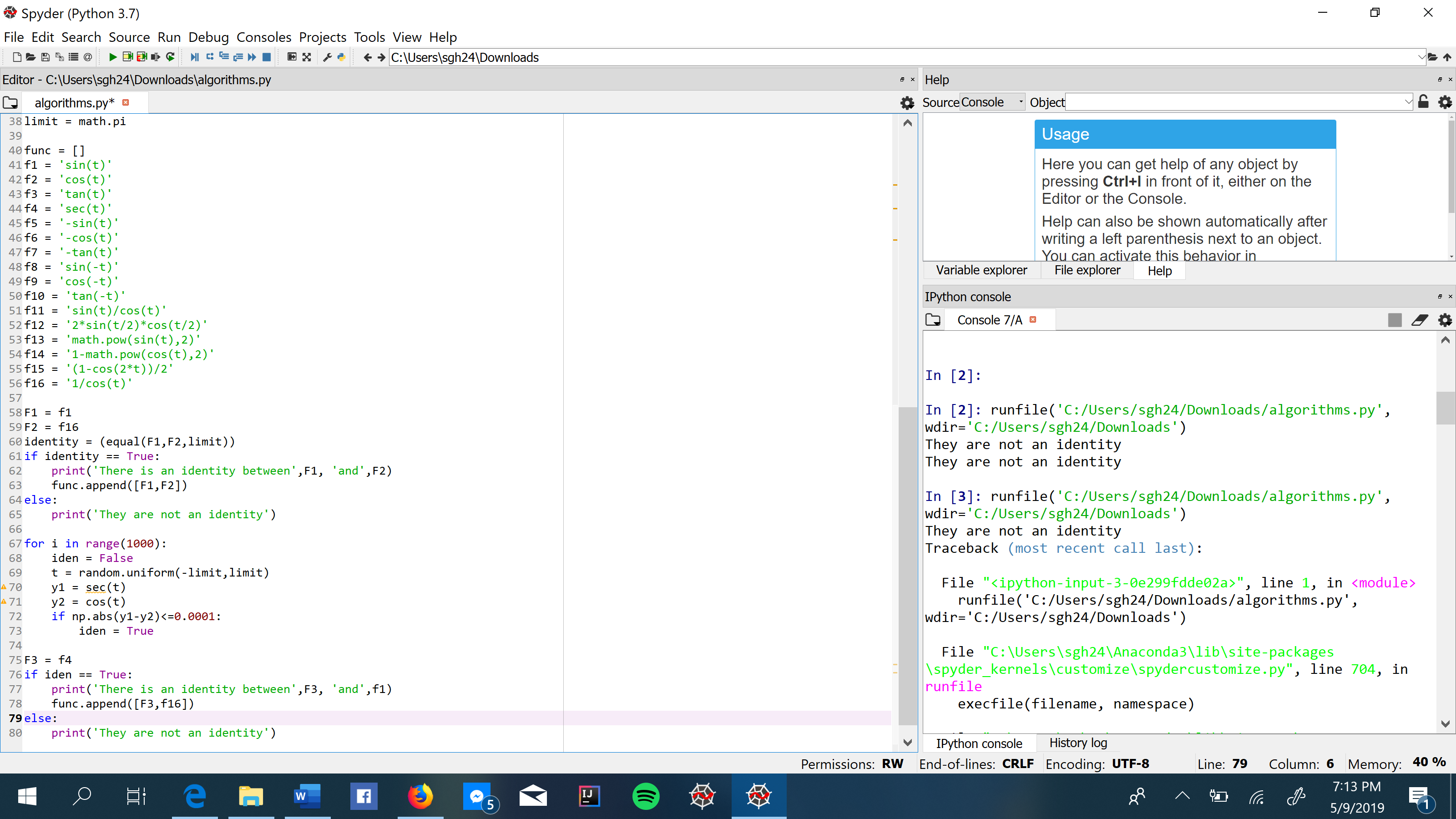
I started with figuring out on how to compute and know the identities of the trigonometric functions. Using the code provided, I added a limit as a parameter in the equal method, and used another utility of random, which is known as uniform, that allows a boundary between to floating numbers, which suits the situation of trigonometric functions due that pi was the given limit. Then I put two types or ways to solve the identity issue. For the first type, I did a for loop that compared each of the elements with one another to know if they were equal, and if they were, I appended them to a list so at the end I could display to the user what are the identities, but the running time was 2.696909189224243 which was longer than type 2 and the list included the identities twice. For type 2, I compared each one of them, and after comparing all of them, I did the list, appended the ones that the program showed as an identity and the running time was 0.06044340133666992. The pros on using type 2 is that it takes less running time and does not repeat the identities twice, but the cons are that you have to do each comparations, which on type one the for loop take care of this. Both of them were correct and solved the identity issue by using the randomization algorithm.

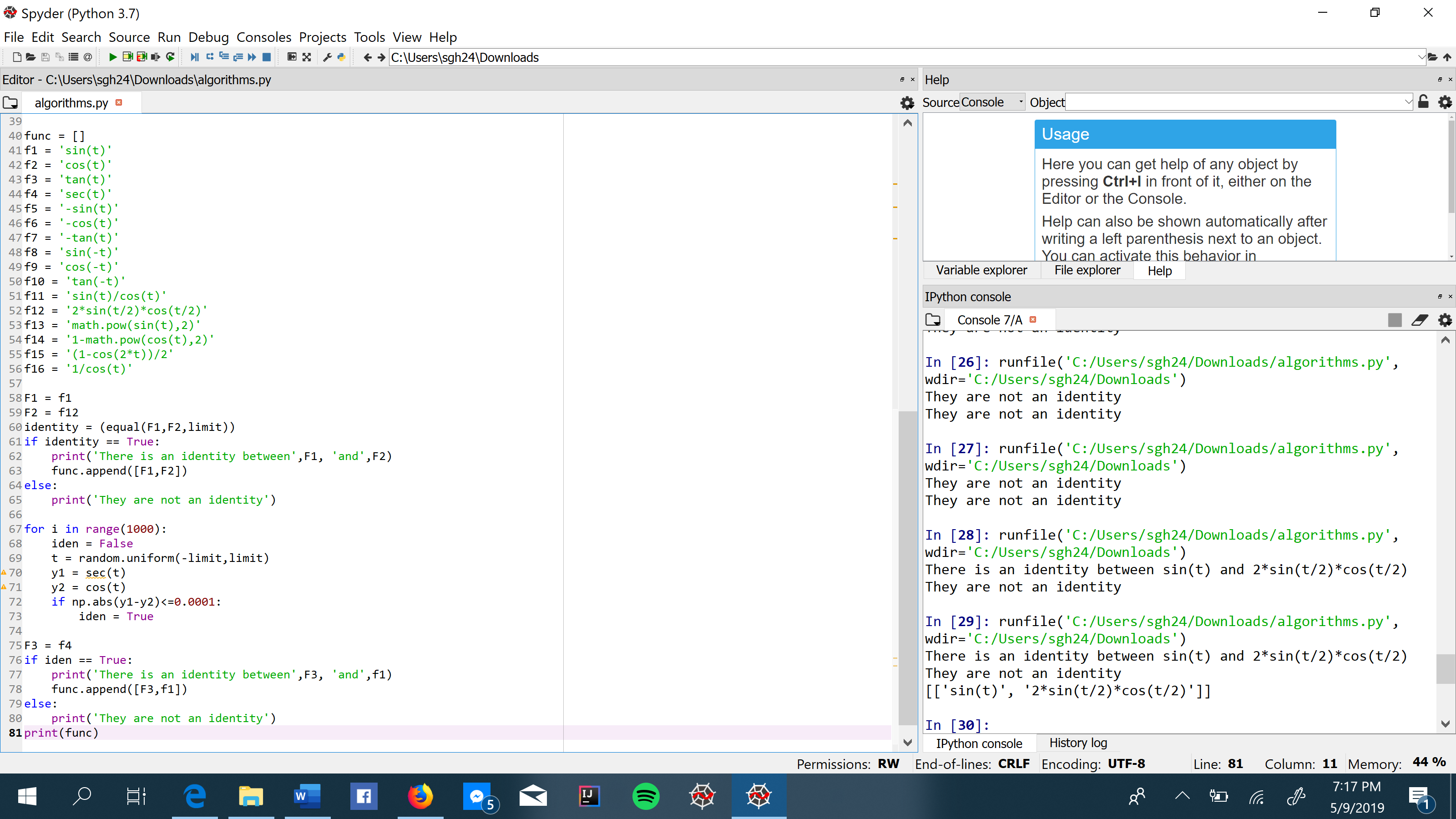
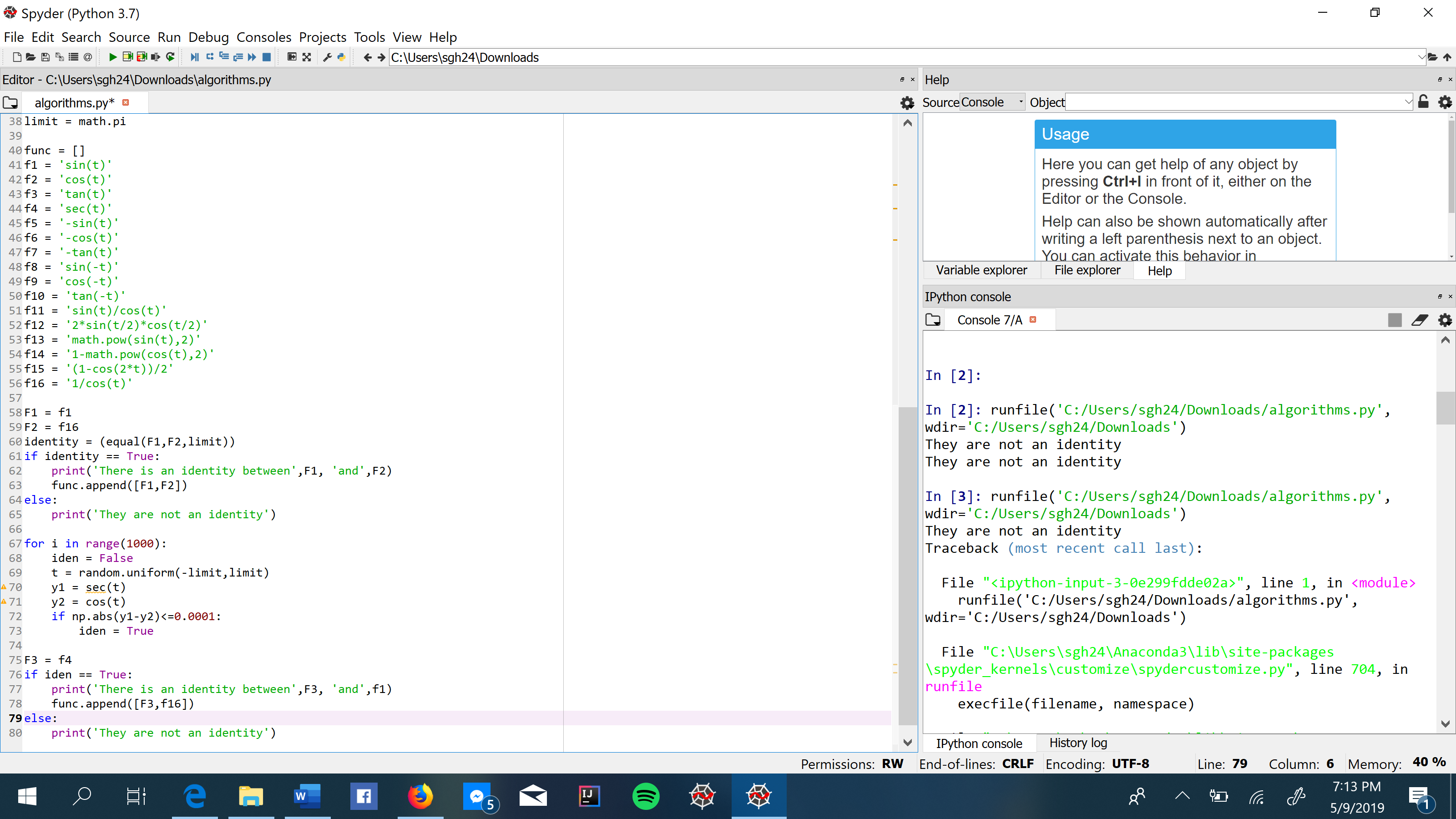
For the partition problem, I based my code on the code provided that did subset sum. In order for a partition to exist, both subsets needed to sum exactly the half of the total sum of the given set of numbers, so by knowing this if the total sum was odd, a partition was not possible, but even if it was not, there was also the possibility of not being a partition. For this problem, I created a method calles partition that had as parameter, the set of numbers, the total sum, a zero which I called first, a last, goal1 and goal2. I included the total sum to know it the sum was even or odd, and if it was odd, it returned false. I added first as parameter to start with beginning of the list, and last to start with the end of the list. I used first and last because I needed to return two subsets. Goal1 and goal2 were the half of the total sum which each subset needed to add up to. My if statements were that if my goal1 and my goal2 were equal to zero, then it would return True and the two subests, my other if statement was if my first was greater than the length of the list of integers or if last was less than zero, then it would return False. Then I recursively added to first and subtracted to last, goal1 and goal2. If it was true, then it would append to my first subset the last element and on the second subset the first element, and by recursion it would add all of them to each subset list and then returning the subsets.

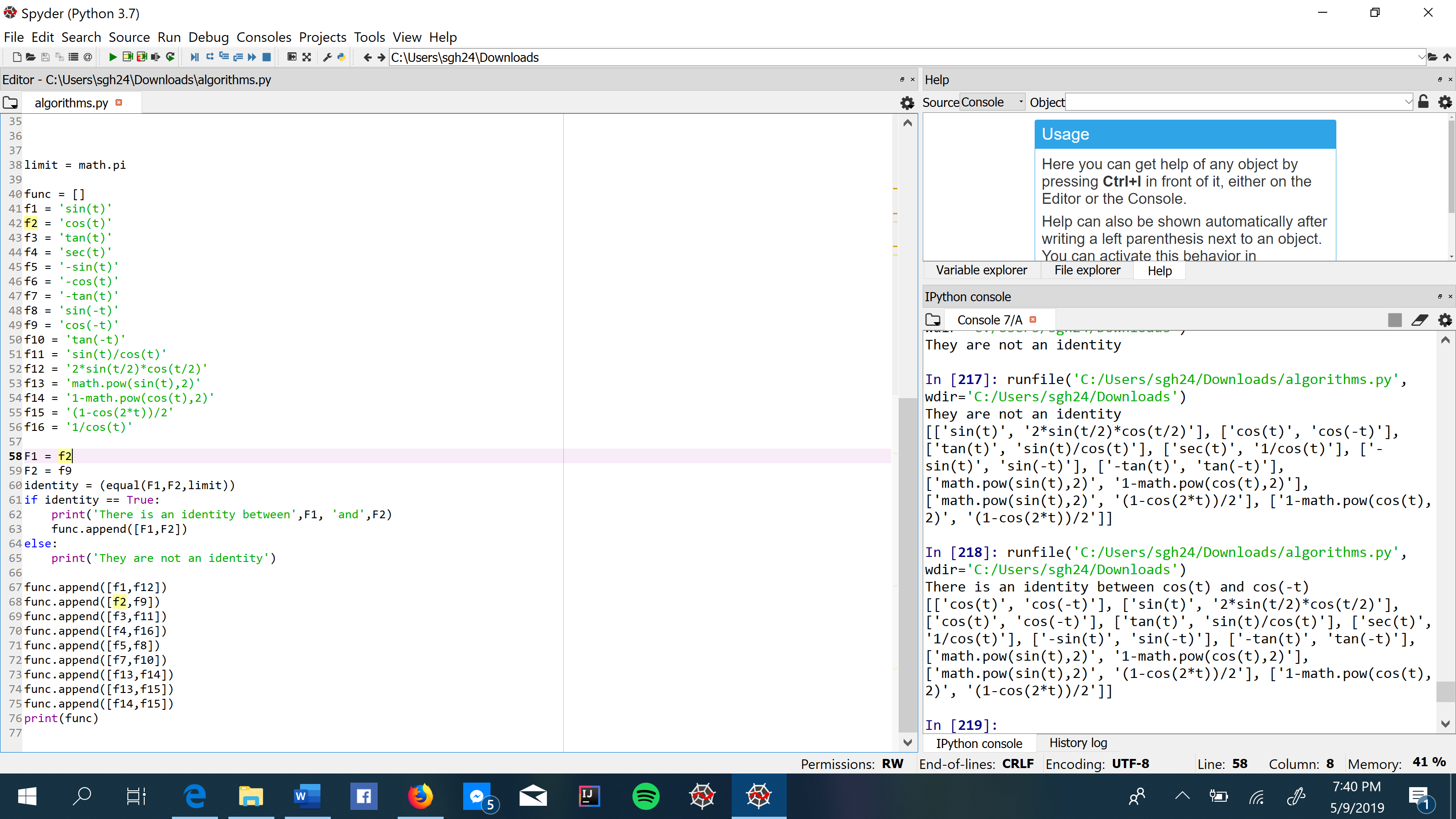
|  |  |
| --- | --- |
| Randomization Type | Running Times |
| Type 1 | 2.696909189224243 seconds |
| Type 2 | 0.06044340133666992 seconds |

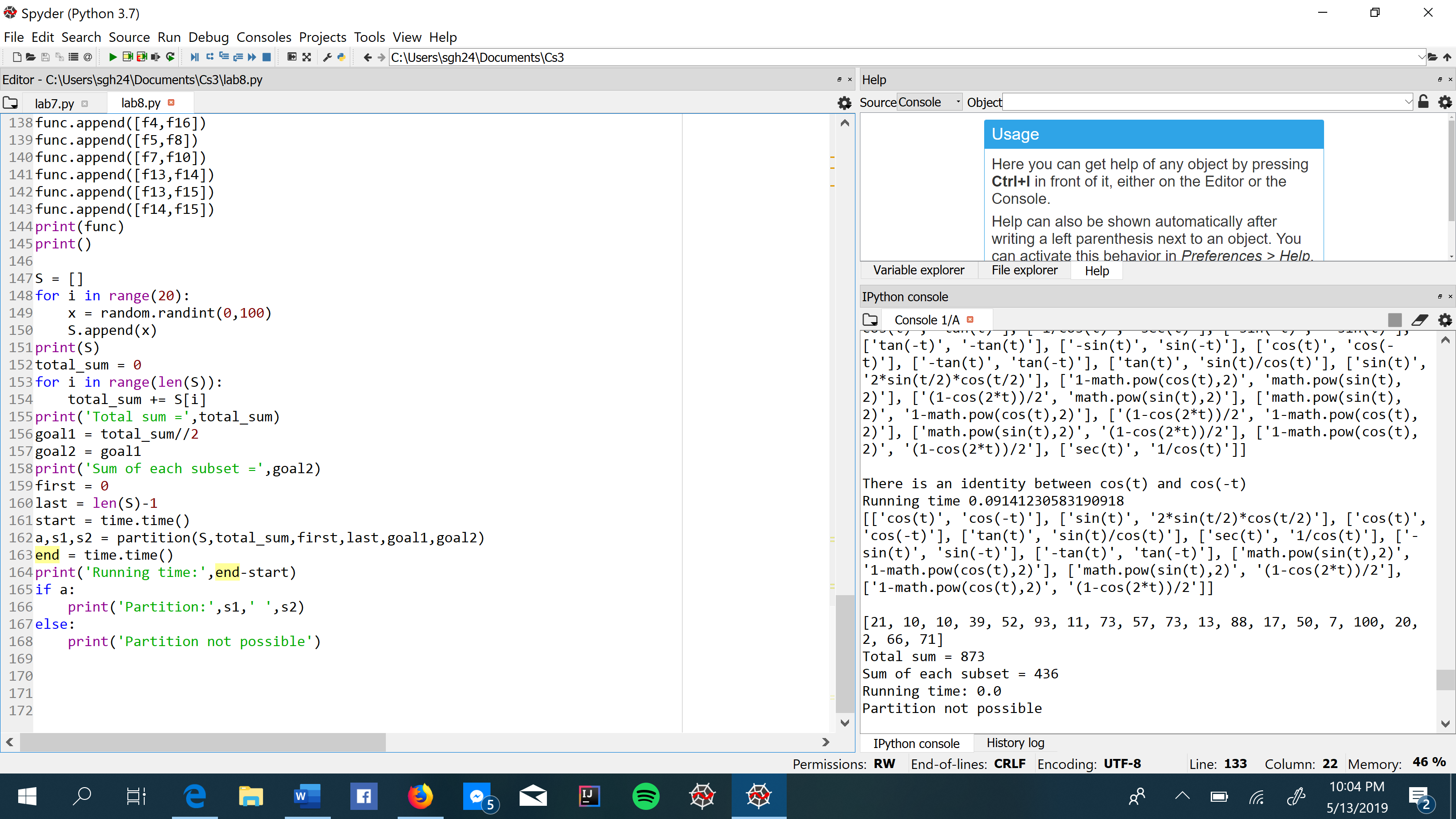
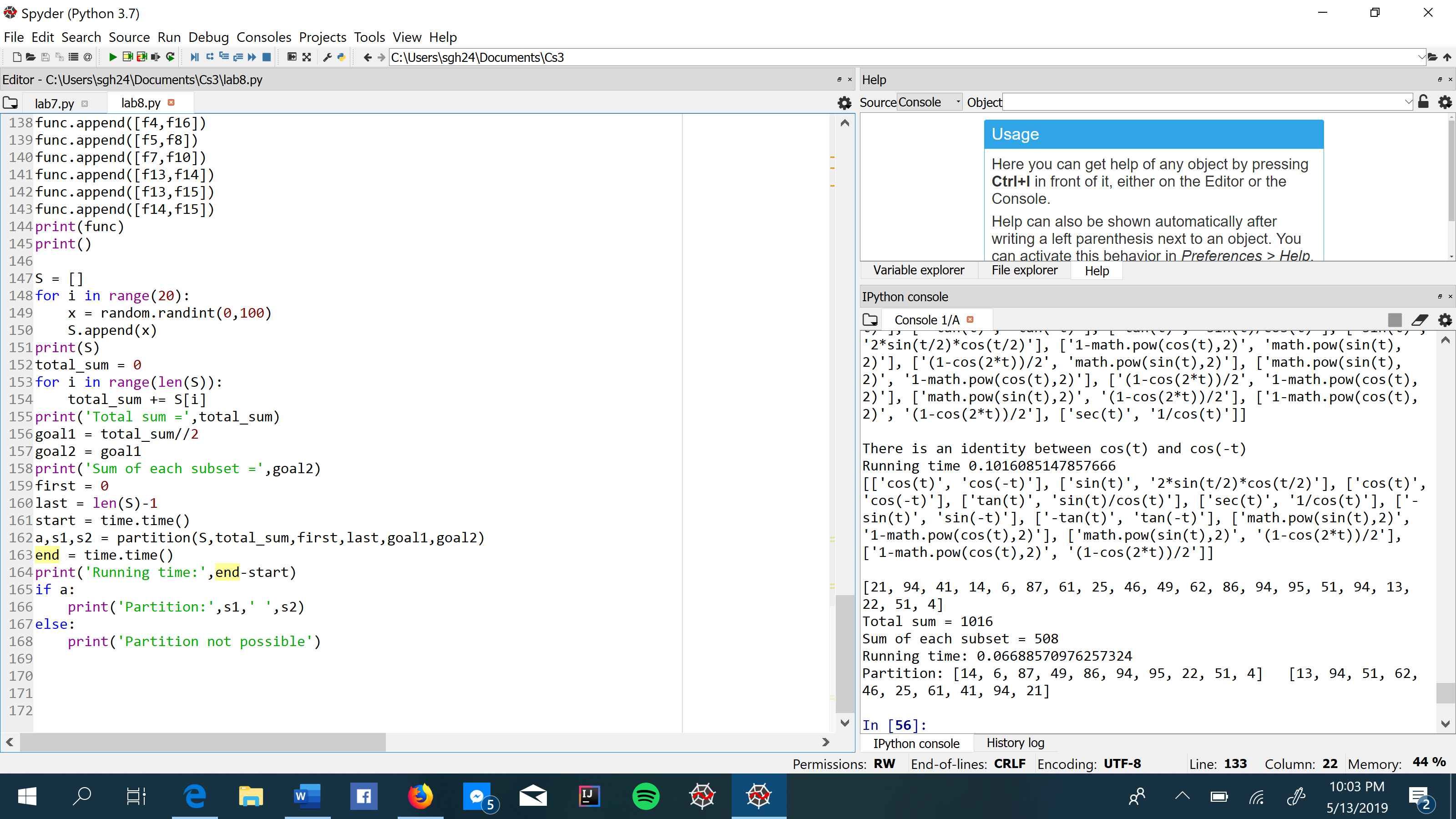
|  |  |
| --- | --- |
| Size of set of integers | Running Time |
| 20 | 0.01772308349609375 |
| 30 | 0.050366878509521484 |
| 40 | 0.7498023509979248 |
| 50 | 3.3780126571655273 |
| 60 | 10.53831434249878 |

**Screenshots:**









**Conclusion:**

In this lab, I learned how useful randomization algorithms can be to the point that this program can help on my math homework. Also, I learned how backtracking works and how useful can be when exhaustive search is needed. I hope that I will keep on getting better and improve my programming skills.

**I, Sebastian Gomez, certify that this project is entirely my own work, I wrote, debugged, and tested the code being presented, performed 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.**

**Appendix:**

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

"""

@author: Sebastian Gomez

Course: Data Structure 2302

Assignment: Lab 8

Instructor: Olac Fuentes

T.A: Anindita Nath and Maliheh Zargaran

Purpose: Learn how randomizing algorithms are useful and how backtracking works

"""

import random

from math import \*

import numpy as np

import math

from mpmath import \*

import time

def equal(f1, f2,limit,tries=1000,tolerance=0.0001):

for i in range(tries):

t = random.uniform(-limit,limit)

y1 = eval(f1)

y2 = eval(f2)

if np.abs(y1-y2)>tolerance:

return False

return True

def partition(S,total\_sum,first,last,goal1,goal2):

if total\_sum%2!=0:

return False,[],[]

if goal1==0 and goal2==0:

return True,[],[]

if first>len(S)-1 or last<0:

return False,[],[]

if goal1<0 and goal2<0:

return False,[],[]

res,subset1,subset2 = partition(S,total\_sum,first+1,last-1,goal1-S[last],goal2-S[first])

if res:

subset1.append(S[last])

subset2.append(S[first])

return True,subset1,subset2

else:

return partition(S,total\_sum,first+1,last-1,goal1,goal2)

limit = math.pi

func = []

f1 = 'sin(t)'

f2 = 'cos(t)'

f3 = 'tan(t)'

f4 = 'sec(t)'

f5 = '-sin(t)'

f6 = '-cos(t)'

f7 = '-tan(t)'

f8 = 'sin(-t)'

f9 = 'cos(-t)'

f10 = 'tan(-t)'

f11 = 'sin(t)/cos(t)'

f12 = '2\*sin(t/2)\*cos(t/2)'

f13 = 'math.pow(sin(t),2)'

f14 = '1-math.pow(cos(t),2)'

f15 = '(1-cos(2\*t))/2'

f16 = '1/cos(t)'

f = ['sin(t)','cos(t)','tan(t)','sec(t)','-sin(t)','-cos(t)','-tan(t)','sin(-t)','cos(-t)','tan(-t)','sin(t)/cos(t)','2\*sin(t/2)\*cos(t/2)','math.pow(sin(t),2)','1-math.pow(cos(t),2)','(1-cos(2\*t))/2','1/cos(t)']

ff = []

start = time.time()

for i in range(len(f)):

y = f[i]

iden = equal(f1,y,limit)

if iden==True and f1!=f[i]:

ff.append([f1,y])

iden = equal(f2,y,limit)

if iden==True and f2!=f[i]:

ff.append([f2,y])

iden = equal(f3,y,limit)

if iden==True and f3!=f[i]:

ff.append([f3,y])

iden = equal(f4,y,limit)

if iden==True and f4!=f[i]:

ff.append([f4,y])

iden = equal(f5,y,limit)

if iden==True and f5!=f[i]:

ff.append([f5,y])

iden = equal(f6,y,limit)

if iden==True and f6!=f[i]:

ff.append([f6,y])

iden = equal(f7,y,limit)

if iden==True and f7!=f[i]:

ff.append([f7,y])

iden = equal(f8,y,limit)

if iden==True and f8!=f[i]:

ff.append([f8,y])

iden = equal(f9,y,limit)

if iden==True and f9!=f[i]:

ff.append([f9,y])

iden = equal(f10,y,limit)

if iden==True and f10!=f[i]:

ff.append([f10,y])

iden = equal(f11,y,limit)

if iden==True and f11!=f[i]:

ff.append([f11,y])

iden = equal(f12,y,limit)

if iden==True and f12!=f[i]:

ff.append([f12,y])

iden = equal(f13,y,limit)

if iden==True and f13!=f[i]:

ff.append([f13,y])

iden = equal(f14,y,limit)

if iden==True and f14!=f[i]:

ff.append([f14,y])

iden = equal(f15,y,limit)

if iden==True and f15!=f[i]:

ff.append([f15,y])

iden = equal(f16,y,limit)

if iden==True and f16!=f[i]:

ff.append([f16,y])

end = time.time()

print('Running time:',end-start)

print(ff)

print()

F1 = f2

F2 = f9

start = time.time()

identity = (equal(F1,F2,limit))

if identity == True:

print('There is an identity between',F1, 'and',F2)

func.append([F1,F2])

else:

print('They are not an identity')

end = time.time()

print('Running time:',end-start)

func.append([f1,f12])

func.append([f2,f9])

func.append([f3,f11])

func.append([f4,f16])

func.append([f5,f8])

func.append([f7,f10])

func.append([f13,f14])

func.append([f13,f15])

func.append([f14,f15])

print(func)

print()

S = []

for i in range(60):

x = random.randint(0,100)

S.append(x)

print(S)

total\_sum = 0

for i in range(len(S)):

total\_sum += S[i]

print('Total sum =',total\_sum)

goal1 = total\_sum//2

goal2 = goal1

print('Sum of each subset =',goal2)

first = 0

last = len(S)-1

start = time.time()

a,s1,s2 = partition(S,total\_sum,first,last,goal1,goal2)

end = time.time()

print('Running time:',end-start)

if a:

print('Partition:',s1,' ',s2)

else:

print('Partition not possible')