**CIS-298 Intro to Python**

**With Professor Robert Mann**

**HW #6**

**Student: Demetrius Johnson**

**01 March 2023**

**Due: 09 March 2023 at 10pm**

# #Author: Demetrius Johnson

#CIS-298 Intro to Python with Professor Robert Mann

#HW6 - Algorithmic Efficiency

#Remember to copy/paste your code for each question separately.

#Then, follow it with a snippet showing the output.

#Due: Thu March 09, 2023 4:00pm

# #Question 5.1:

#Use a list comprehension to square each odd number in a list. The list is input by a sequence of comma-separated numbers.

#Suppose the following input is supplied to the program:

#1,2,3,4,5,6,7,8,9

#Then, the output should be:

#1,9,25,49,81

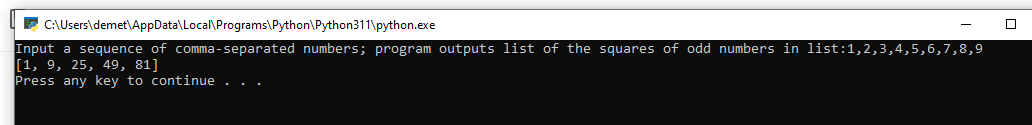
#first, get user input, split it by ',' delimeter into a list of alphanumeric strings

num\_list = input("Input a sequence of comma-separated numbers; program outputs list of the squares of odd numbers in list:").split(',')

#now print the list object returned from the list comprehension:

#cast string as int and square it and add it to list if it is an odd number (there is a remainder after dividing by 2)

print([int(num)\*\*2 for num in num\_list if (int(num) % 2 != 0)])



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

# #Question 6 (optimize hw5 q10):

import time #import time ahead of time so no wasted time on imports on the first call to some of the functions used for timing

#define a timing function that I can use to easily pass in any function to test it and for a desired number of iterations:

def timing\_function(function\_to\_time, num\_iterations=1):

##import time function in order to time our functions

import time

time\_sum = 0

total\_iterations = num\_iterations

print("Number of iterations function will run:", num\_iterations)

while(num\_iterations > 0):

start = time.perf\_counter()

function\_to\_time()

end = time.perf\_counter()

elapsed\_time = end - start

print("\n----Time elapsed for iteration#%d:"% (total\_iterations - num\_iterations + 1),elapsed\_time)

time\_sum += elapsed\_time #track sum of time for all iterations

num\_iterations -= 1 #track iteration value

print( "\n----Total Time: %f----Average Time: %f"%( time\_sum,(time\_sum/total\_iterations) ) )

## #Question 6.1:

### Part A

#Q6.1[10 pts]. Determine how to time programs in Python.

# A) Run 3 tests of each of the first two solutions given above

# and average the results.

##Now, time given solutions, and develop my own solutions to time as well:

#soliution 1:

def even\_nums\_sol\_1():

print(\*[n for n in range(1000, 3001) if all([(int(c) % 2 == 0) for c in str(n)])], sep=',')

#call timing function for solution 1:

timing\_function(even\_nums\_sol\_1,num\_iterations=3)

Graphical user interface, text

Description automatically generated

**Average Time = 0.014637**

#soliution 2:

def even\_nums\_sol\_2():

numbers = []

for i in range(1000, 3001):

is\_even = True

for j in str(i):

if int(j) %2 == 1:

is\_even = False

if is\_even:

numbers.append(i)

print(numbers)

#call timing function for solution 2:

timing\_function(even\_nums\_sol\_2,num\_iterations=3)

Calendar

Description automatically generated with medium confidence

**Average Time = 0.008828**

### Part B

# B) Come up with a solution that is more efficient (faster).

# Run 3 tests on it and record the average.

#my\_solution\_1:

def my\_solution\_1(lower\_range=1000, upper\_range=3000):

lower\_range\_digits = list(map(int, str(lower\_range))) #convert lower\_range into a list of digits

if (lower\_range % 2) == 1:

lower\_range += 1

halved\_search\_list = range(lower\_range, upper\_range, 2)

def check\_all\_digits\_even(digits\_list):

for digit in digits\_list:

if digit % 2 == 1:

return False #found an odd digit

return True #if all digits are even

for number in halved\_search\_list:

if check\_all\_digits\_even( list(map(int, str(number))) ):

print(number,end=',')

print() #print newline at end of function for cleaner output

#call timing function for my\_solution\_1:

print("\n---------TIMING FOR MY\_SOLUTION\_1:")

timing\_function(my\_solution\_1,num\_iterations=3)

#timing\_function(my\_solution\_1,num\_iterations=3)A picture containing timeline

Description automatically generated

### Part C

# C) Supply your faster algorithm and the average result of all three algorithms.

#time all algorithms 3 times:

solution\_1\_avg = timing\_function(even\_nums\_sol\_1,num\_iterations=3)

solution\_2\_avg = timing\_function(even\_nums\_sol\_2,num\_iterations=3)

my\_solution\_1\_avg = timing\_function(my\_solution\_1,num\_iterations=3)

print('\n\*\*\*Average for all 3 functions = ',(solution\_1\_avg + solution\_2\_avg + my\_solution\_1\_avg) / 3)

Graphical user interface, application

Description automatically generated

## #Question 6.22:

#Q6.22[50]. Deep dive into logically cutting corners in a solution through analysis.

#Find a solution to this problem that executes in under 1 second (can it be even faster?) without hardcoding the answer.

#Displaying values takes a long time so remove most output statements once you’re getting close.

#I'll spare you the long version of the question.

#Simply do this: find the price of four items who's sum and who's product both equal 7.11. So, a+b+c+d == a\*b\*c\*d == 7.11.

#There may be more than one answer. I originally solved this problem 37 years ago (in C). My initial algorithm ran for over 12 hours.

#My deep dive analysis produced one that solved it in about 1 second. Supply your final algorithm and how long it takes to solve the problem.

#Find the price of four items who's sum and who's product both equal to 7.11.

#So, a+b+c+d == a\*b\*c\*d == 7.11

#I found from my analysis that using range of number from [5,0) yields a fast result since solutions seem to be very small numbers.

#thus, that is why I make default upper and lower range to be 5, 0 respectively. Precision is 2 by default because of problem requirements.

def mult\_eq\_sum\_FourNums(linear\_combination\_val=7.11, precision=2, upper\_range=5, lower\_range=0):

#I will adapt my random number generator code from hw4 question 3 to generate 2 random numbers

import random

def get\_randmon\_num(positive\_range=100, negative\_range=100):

#random() function returns a number between 0 and 1

#Multiply 100\*random() and subtract 100\*random() in order to make the values have a range of [-100, 100]

#Return the random number if it is nonzero, else return 1 --> thus making our range [-100, 100, exclude 0]

random\_num = positive\_range \* random.random() - negative\_range \* random.random()

return random\_num if random\_num else 1

#need to find a solution for items a and b

a = 0

b = 0

c = 0

d = 0

#use these just for metric purposes

num\_iterations\_partial = 0

num\_iterations\_full = 0

num\_iterations\_total = 0

#Now use this loop to find a solution for a and b that does not include the complex number solutions:

while (a <= 0) or (b <= 0) or \

( round(a\*b\*c\*d,precision) != round(linear\_combination\_val,precision) ) or \

( round(a+b+c+d,precision) != round(linear\_combination\_val,precision)

):

#So after further analysis, I realize I can simplify the problem!

#let c and d be any (random) number, then the problem is simplified from:

#now generate 2 random values (we will use range (0,100]) that will be used)

#set negative range = 0 since prices of items must be nonzero

c = get\_randmon\_num(positive\_range=upper\_range,negative\_range=lower\_range)

d = get\_randmon\_num(positive\_range=upper\_range,negative\_range=lower\_range)

#abcd=7.11 and a+b+c+d=7.11

#to ab\*c\*d = 7.11 and a+b+c+d = 7.11, where c and d are known (chosen/selected).

#which simplifies to: ab=7.11/cd and a+b=7.11-(c+d)

#thus we simply need to solve a system of equations for terms a and b:

#EQUATION\_1: a = (7.11/cd)/b,

#EQUATION\_2: a = 7.11 - (c+d) - b

#--> (7.11/cd)/b = 7.11 - (c+d) - b

#--> (7.11/cd) = 7.11b - (c+d)b - b^2

#--> -(7.11/cd) = -7.11b + (c+d)b + b^2

#--> b^2 + (c+d)b - 7.11b = -(7.11/cd)

#--> b^2 + (c+d-7.11)b + (7.11/cd) = 0

#now solve using quadratic formula: x = (-c2 (+/-) sqrt[c2^2 -4\*c1\*C]) / 2\*c1,

#where c\_i is a coefficient of ith-power term, C is a constant

c1 = 1

c2 = (c+d-linear\_combination\_val)

C = (linear\_combination\_val/(c\*d))

Discriminant = c2\*\*2 - 4\*c1\*C

#use quadratic formula, which yields 2 possible solutions for b:

# solution 1 for b: ( - (c+d-7.11) (+) sqrt[(c+d-7.11)^2 - 4\*1\*(7.11/cd)] ) / (2\*1)

# solution 2 for b: ( - (c+d-7.11) (-) sqrt[(c+d-7.11)^2 - 4\*1\*(7.11/cd)] ) / (2\*1)

#once we use one of the solutions for b, we simply plug in b's value to any one of our system equations to solve for a

#We first need to do a special check on the discriminant D = c2^2-4\*c1\*C :

#if D < 0, then sqrt(c2^2-4\*c1\*C) will be a complex number since it will take sqrt of a negative number (complex solution);

#if D > 0, then there are two real roots (thus 2 real number solution)

#if D == 0, then there is one real root (thus one real number solution)

#A simpler way of saying this is that if c2^2 < 4\*c1\*C, then sqrt(c2^2-4\*c1\*C) will be a complex number;

#thus, we need to check for this and skip this loop since values chosen for c and d will produce complex solutions:

if Discriminant < 0:

num\_iterations\_partial += 1

continue #move to next loop and try with 2 new values for c and d

#otherwise, continue the algorithm (I will use solution 1 equation to solve for b):

b = ( -(c2) + (Discriminant)\*\*(1/2) ) / (2\*c1)

a = linear\_combination\_val - (c+d) - b

num\_iterations\_full += 1 #full iteration completed...

print('Iteration\_#\_%d:'%(num\_iterations\_partial + num\_iterations\_full),

'a=%f'%a,

'b=%f'%b,

'c=%f -> randomnly selected from range: (%f,%f]'%(c,lower\_range,upper\_range),

'd=%f -> randomnly selected from range: (%f,%f]'%(d,lower\_range,upper\_range), sep='\n' )

print('Is a+b+c+d == a\*b\*c\*d?: ', a+b+c+d, '==', a\*b\*c\*d, '?')

##end of WHILE loop##

#set total number of iterations metric

num\_iterations\_total = num\_iterations\_partial + num\_iterations\_full

#print final results:

print('\n\n\*\*SOLUTION FOUND WITH ALL POSTIVE VALUES\*\*:', 'a=%f'%a,'b=%f'%b,'c=%f'%c,'d=%f'%d,

'partial\_iterations:%d (due to Discriminant < 0)'%num\_iterations\_partial,

'full\_iterations:%d'%num\_iterations\_full,

'total\_iterations:%d'%num\_iterations\_total, sep='\n' )

print('Is a+b+c+d == a\*b\*c\*d?: ', a+b+c+d, '==', a\*b\*c\*d, '? --> YES, solution found (rounded to %d decimal places)!'%precision)

##end of mult\_eq\_sum\_FourNums() function##

##now time my function

import time

sum\_elapsed\_times = 0

num\_runs = 3

while num\_runs > 0:

start = time.perf\_counter()

mult\_eq\_sum\_FourNums(linear\_combination\_val=7.11, precision=1, upper\_range=7.11, lower\_range=0)

end = time.perf\_counter()

elapsed\_time = end - start

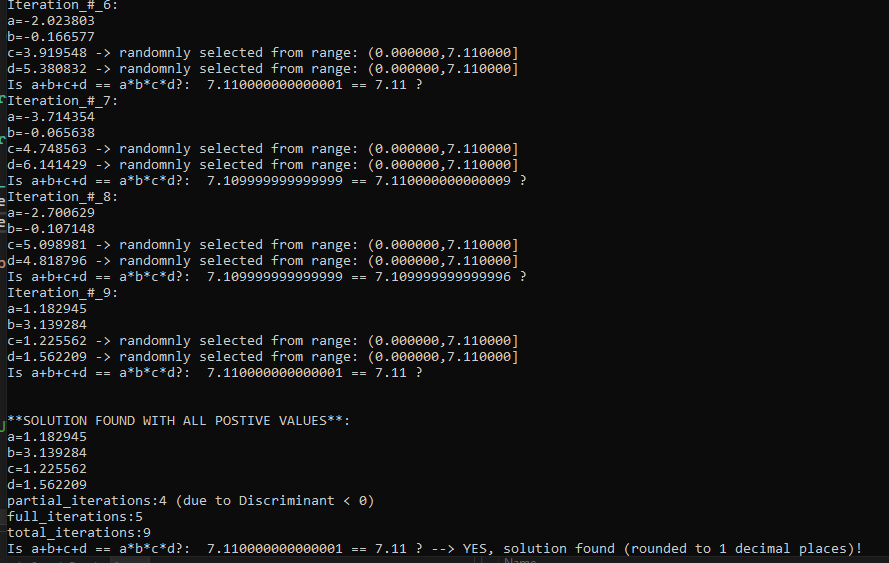
sum\_elapsed\_times += elapsed\_time

num\_runs -= 1

print("time elapsed:",elapsed\_time)

print("AVERAGE TIME FOR 3 RUNS:", (sum\_elapsed\_times / 3))

### EXAMPLE OF SOLUTION:



### AVERAGE TIME FOR 3 RUNS:

Text

Description automatically generated

# References:

* Information on ***cProfile (***or ***profile)*** library for timing algorithms and all function calls made.
  + <https://docs.python.org/3/library/profile.html>
* The ***timeit*** library used to measure small snippets of code.
  + <https://docs.python.org/3/library/timeit.html#:~:text=This%20module%20provides%20a%20simple,traps%20for%20measuring%20execution%20times>.
* ***time*** library which both the ***cProfile*** *and* ***timeit*** libraries depend on
  + <https://docs.python.org/3/library/time.html>