See projectRun.py

import time  
import math  
from functools import reduce  
from multiprocessing import Pool  
  
'''  
REQUIREMENTS FOR: COUNTING THE TOTAL NUMBER OF CHARACTERS  
\* know count of each unique (ALREADY DONE)  
\* return sum of these counts  
  
PROBABILITY DISTRIBUTION:  
\* probability p of some character c (pc) is the NUMBER OF OCCURRENCES OF C divided by TOTAL CHARACTERS IN DATA STREAM  
\*\* so pc = nc / total c  
\* probability distribution is the probability of EACH CHARACTER in data stream  
'''  
  
'''  
REQUIREMENTS FOR: A FUNCTIONAL PARADIGM  
\* need to use map/reduce  
\* functional paradigms use CONSTANTS outside their scope; doesn't rely on mutable variables -- passes returned results  
\* from function to function   
\*\* PURE functional programs only depend on their input (no internal state)  
'''  
  
  
def probabilities(count):  
 global MY\_TOTAL\_CHARACTERS  
 return count / MY\_TOTAL\_CHARACTERS  
  
  
def file\_count(char):  
 global FILE\_STRING  
 return FILE\_STRING.count(char)  
  
  
# lambda sum for reduce --------------------------------  
myTotal = lambda x, y: x + y  
  
MY\_TEXT\_FILE = open("C:/Users/Mikaela/Documents/Spring2020/CS441 Progrm LangDes&Imp/Final Project/WarAndPeace.txt",  
 "r", encoding="utf8")  
  
FILE\_STRING = MY\_TEXT\_FILE.read()  
MY\_TOTAL\_CHARACTERS = len(FILE\_STRING)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 startClock = time.perf\_counter\_ns() # EXCLUDE FILE I/O FOR THE TIMER --------------------------  
 MY\_WORKER\_POOL = Pool(processes=60) # THIS IS FOR CONCURRENCY ------- NUMBER IS NUMBER OF THREADS  
 # USE MY\_WORKER\_POOL.MAP INSTEAD OF MAP --- NEED 2, 4, 8, 16, 32, AND 64 THREADS; NEED TO RUN 3 TIMES FOR AVERAGE  
  
 MY\_UNIQUES\_TUPLE = set(x for x in FILE\_STRING) # USED FOR SINGLES  
 # need to pass PAIRS and TRIPLES to count new values for NC -- instead of count char, count pairs, triples...  
 MY\_PAIRS\_TUPLE = set(FILE\_STRING[i: i + 2] for i in range(0, len(FILE\_STRING), 2)) # USED FOR PAIRS  
 MY\_TRIPLES\_TUPLE = set(FILE\_STRING[i: i + 3] for i in range(0, len(FILE\_STRING), 3)) # USED FOR TRIPLES  
  
 # COUNTS SINGLES  
 NUMBER\_OF\_TIMES\_EACH\_CHARACTER\_C\_OCCURS = MY\_WORKER\_POOL.map(file\_count, MY\_UNIQUES\_TUPLE)  
 # print(tuple(NUMBER\_OF\_TIMES\_EACH\_CHARACTER\_C\_OCCURS))  
  
 NUMBER\_OF\_TIMES\_EACH\_PAIR\_OCCURS = MY\_WORKER\_POOL.map(file\_count, MY\_PAIRS\_TUPLE) # PAIRS  
 NUMBER\_OF\_TIMES\_EACH\_TRIPLE\_OCCURS = MY\_WORKER\_POOL.map(file\_count,  
 MY\_TRIPLES\_TUPLE) # TRIPLES  
  
 # MY\_TOTAL\_CHARACTERS = reduce(myTotal, NUMBER\_OF\_TIMES\_EACH\_CHARACTER\_C\_OCCURS) # SPENT ITERATOR  
 # print("TOTAL NUMBER OF CHARACTERS: " + str(MY\_TOTAL\_CHARACTERS) + " (used to solve for pc)")  
  
 # iterator is spent to find MY\_TOTAL\_CHARACTERS, need to remake  
 # reset iterator  
 # NUMBER\_OF\_TIMES\_EACH\_CHARACTER\_C\_OCCURS = MY\_WORKER\_POOL.map(file\_count, MY\_UNIQUES\_TUPLE)  
  
 MY\_PROBABILITIES\_OF\_EACH\_CHARACTER = MY\_WORKER\_POOL.map(probabilities,  
 NUMBER\_OF\_TIMES\_EACH\_CHARACTER\_C\_OCCURS)  
 # SPENT ITERATOR AGAIN  
  
 # print(tuple(MY\_PROBABILITIES\_OF\_EACH\_CHARACTER))  
  
 MY\_PROBABILITIES\_OF\_EACH\_PAIR = MY\_WORKER\_POOL.map(probabilities,  
 NUMBER\_OF\_TIMES\_EACH\_PAIR\_OCCURS) # SPENT  
 MY\_PROBABILITIES\_OF\_EACH\_TRIPLE = MY\_WORKER\_POOL.map(probabilities,  
 NUMBER\_OF\_TIMES\_EACH\_TRIPLE\_OCCURS) # SPENT  
  
 # RESET NC AGAIN  
 NUMBER\_OF\_TIMES\_EACH\_CHARACTER\_C\_OCCURS = MY\_WORKER\_POOL.map(file\_count, MY\_UNIQUES\_TUPLE)  
 NUMBER\_OF\_TIMES\_EACH\_PAIR\_OCCURS = MY\_WORKER\_POOL.map(file\_count, MY\_PAIRS\_TUPLE)  
 NUMBER\_OF\_TIMES\_EACH\_TRIPLE\_OCCURS = MY\_WORKER\_POOL.map(file\_count, MY\_TRIPLES\_TUPLE)  
  
 # map this function: (nc)(−pc)lg(pc) to each NC  
 MY\_INFORMATION\_OF\_EACH\_CHARACTER = map((lambda nc, pc: round(nc \* (-pc) \* math.log(pc, 2), 3)),  
 NUMBER\_OF\_TIMES\_EACH\_CHARACTER\_C\_OCCURS,  
 MY\_PROBABILITIES\_OF\_EACH\_CHARACTER)  
 MY\_INFORMATION\_OF\_EACH\_PAIR = map((lambda nc, pc: round(nc \* (-pc) \* math.log(pc, 2), 3)),  
 NUMBER\_OF\_TIMES\_EACH\_PAIR\_OCCURS,  
 MY\_PROBABILITIES\_OF\_EACH\_PAIR)  
 MY\_INFORMATION\_OF\_EACH\_TRIPLE = map((lambda nc, pc: round(nc \* (-pc) \* math.log(pc, 2), 3)),  
 NUMBER\_OF\_TIMES\_EACH\_TRIPLE\_OCCURS,  
 MY\_PROBABILITIES\_OF\_EACH\_TRIPLE)  
  
 # get the summation  
 MY\_TOTAL\_INFORMATION\_IN\_STREAM = round(reduce(myTotal, MY\_INFORMATION\_OF\_EACH\_CHARACTER), 3)  
 MY\_TOTAL\_INFORMATION\_PAIRS = round(reduce(myTotal, MY\_INFORMATION\_OF\_EACH\_PAIR), 3)  
 MY\_TOTAL\_INFORMATION\_TRIPLES = round(reduce(myTotal, MY\_INFORMATION\_OF\_EACH\_TRIPLE), 3)  
  
 RUN\_TIME = ((time.perf\_counter\_ns()) - startClock) # END TIME - START TIME to get duration  
 print(str(RUN\_TIME) + " nanoseconds")  
 print(str(RUN\_TIME / 1000) + " microseconds (μs)")  
 print(str(RUN\_TIME / 1000000000) + " seconds") # to return seconds  
 # 2 decimals w/ str(round(answer, 2))  
 print(str(round(RUN\_TIME / 1000000000, 2)) + " seconds (rounded)")  
  
 # TOTAL INFORMATION IN THE FILE  
 print("INFORMATION IN DATA STREAM: " + str(MY\_TOTAL\_INFORMATION\_IN\_STREAM) + " (SINGLES)")  
 print("INFORMATION IN DATA STREAM: " + str(MY\_TOTAL\_INFORMATION\_PAIRS) + " (PAIRS)")  
 print("INFORMATION IN DATA STREAM: " + str(MY\_TOTAL\_INFORMATION\_TRIPLES) + " (TRIPLES)")