#This script is part of supplementary documents of "Impact of Introns and Homing Endonucleases on Structural Mitogenome Shaping in Hypocreales"

#submitted to Frontiers in Microbiology, section Fungi and Their Interactions

#Manuscript ID: 531057

#Authors: Paula Fonseca, Fernanda Badotti, Ruth De-Paula, Daniel Araújo, Dener Eduardo Bortolini, Luiz-Eduardo Del-Bem, Vasco Ariston De Carvalho Azevedo,

#Bertram Brenig, Eric Roberto Guimarães Rocha Aguiar, Aristóteles Góes-Neto

#This script uses the NCBI API to make a query and retrieve data from GenBank with all gene annotation of target species

#Using a text file as input, the script can retrieve data from multiples species at time

#The result is a '.cds' file, with name, ID, size of genome, start, end positions and the name of all genes

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Run the code in Python 3+ #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# -\*- Coding: UTF-8 -\*-

#coding: utf-8

import sys

import urllib.request

import re

import os.path

from os import path

#This function reads the target(s) id(s) specie(s) from a 'txt' file

def readIDs(fileName\_txt):

 ids\_file=open(fileName\_txt,'r')

 query\_IDs = []

 for line in ids\_file:

 #the '[accn]' substring is added as requirement by NCBI API

 query\_IDs.append(line.rstrip()+"[accn]")

 ids\_file.close()

 return query\_IDs

#This function will try to retrieve the data of a target specie using NCBI API. The result is a xml string with all data.

def getXMLNCBI(str\_ID):

 urlBase = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/"

 #create URL for esearch

 url = urlBase+"esearch.fcgi?db=nuccore&term="+str\_ID+"&usehistory=y"

 f = urllib.request.urlopen(url)

 #Read xml from url

 xml = f.read()

 strXml=xml.decode("utf-8")

 #If WebEnv e QueryKey exists in this firstxml, fetch the URL query

 objRe = re.search('<WebEnv>(\S+)<\/WebEnv>',strXml)

 webEnv = objRe.group()

 webEnv = webEnv[8:len(webEnv)-9]

 objRe = re.search('<QueryKey>(\d+)<\/QueryKey>',strXml)

 queryKey = objRe.group()

 queryKey = queryKey[10:len(queryKey)-11]

 #URL efetch

 url = urlBase+"efetch.fcgi?db=nuccore&query\_key="+queryKey+"&WebEnv="+webEnv+"&rettype=gb&retmode=xml"

 f = urllib.request.urlopen(url)

 xml\_esearch\_bin = f.read()

 xml\_esearch = xml\_esearch\_bin.decode("utf-8")

 return xml\_esearch

#This function generates xml and cds files with the data retrieved from NCBI. The CDS file will contain all the annotated genes within the genome of a specie

def generateXMLCDS(item, xml\_esearch, key\_words):

 genome\_ID=""

 genome\_size=0

 genome\_size\_CDS=0

 #To treat gene overlapping, the array cds\_vector will store the nucleotides positons that are part of a coding region

 #If a nucleotide is part of CDS, then its position on cds\_vector will be '1', while the positions with '0' will represent the nucleotides of non coding region

 cds\_vector=[]

 #The index\_key\_words is a counter that controls the exploration of the ordered indendation and alignment in the xml format, retrieving the gene data when found

 index\_key\_words=0

 #Open output files .xml and .cds

 output\_xml\_file=open(item[:len(item)-6]+".xml",'w')

 output\_cds\_file=open(item[:len(item)-6]+".cds","w")

 for line in xml\_esearch:

 output\_xml\_file.write(line+"\n")

 #The command below search for the key\_word indicated by index\_key\_words in the present line of the xml

 id\_str\_found=line.find(key\_words[index\_key\_words])

 if(id\_str\_found!=-1):

 #if it is the first index\_key\_word, get the genome ID and set to next key\_word

 if (index\_key\_words==0):

 genome\_ID=line[id\_str\_found+len(key\_words[index\_key\_words]):line.find("<",id\_str\_found+1)]

 index\_key\_words=index\_key\_words+1

 #If it is the second one, get genome total size and set to next key\_word

 elif (index\_key\_words==1):

 genome\_size=int(line[id\_str\_found+len(key\_words[index\_key\_words]):line.find("<",id\_str\_found+1)])

 #Instantiate cds\_vector with size + 1 of the whole genome. The 0 position will not be used

 cds\_vector = [0]\*(genome\_size+1)

 genome\_size\_CDS=0

 index\_key\_words=index\_key\_words+1

 #In the third key\_word, get specie name header, write in screen and cds file and set to next key\_word

 elif (index\_key\_words==2):

 output\_cds\_file.write(line[id\_str\_found+len(key\_words[index\_key\_words]):line.find("<",id\_str\_found+1)]+"\n")

 output\_cds\_file.write("Genome ID: "+genome\_ID+"\n")

 output\_cds\_file.write("Genome size: "+str(genome\_size)+"\n")

 output\_cds\_file.write("Genes:\n")

 print(line[id\_str\_found+len(key\_words[index\_key\_words]):line.find("<",id\_str\_found+1)])

 print("Genome ID: "+genome\_ID)

 print("Genome size: "+str(genome\_size))

 index\_key\_words=index\_key\_words+1

 #Here we get the information if the next data is part of a gene and go further into the xml with another key\_word

 elif (index\_key\_words==3):

 index\_key\_words=index\_key\_words+1

 #In the fifth one, we get the string with the start and end positions of the gene, calling a function that print the values on screen and cds file,

 #besides marking the nuclotides positions of the gene on cds\_vector

 elif (index\_key\_words==4):

 #Treat a Join if necessary

 if (line.find("join")==-1):

 rangeCDS=re.sub('[^0-9.]','',line)

 write\_start\_end\_gene(rangeCDS,output\_cds\_file,cds\_vector)

 else:

 output\_cds\_file.write("\n\n")

 rangeCDS=re.sub('[^0-9.,]','',line)

 rangeCDS1=rangeCDS[:rangeCDS.find(",")]

 write\_start\_end\_gene(rangeCDS1,output\_cds\_file,cds\_vector)

 print()

 rangeCDS2=rangeCDS[rangeCDS.find(",")+1:]

 write\_start\_end\_gene(rangeCDS2,output\_cds\_file,cds\_vector)

 index\_key\_words=index\_key\_words+1

 #We go further into the xml indendation

 elif (index\_key\_words==5):

 index\_key\_words=index\_key\_words+1

 #And get the gene name, printing on screen and cds file

 elif (index\_key\_words==6):

 gene\_name=line[line.find("<GBQualifier\_value>")+19:line.find("</GBQualifier\_value>",18)]

 output\_cds\_file.write("#"+gene\_name+"\n")

 print(" ("+str(gene\_name)+")")

 #Then we retrocede 3 positions in index\_key\_value to look for another genes

 index\_key\_words=index\_key\_words-3

 #Get the number of nucleotides in coding regions

 genome\_size\_CDS=sum(cds\_vector)

 print("Sum of nucleotides in the coding regions (CDS) of the genome ID= "+genome\_ID+": "+str(genome\_size\_CDS)+" of "+str(genome\_size)+" nucleotides ("+str(round(genome\_size\_CDS\*100/genome\_size,2))+"%)")

 output\_cds\_file.write("Sum of nucleotides in the coding regions (CDS) of the genome: "+str(genome\_size\_CDS)+" of "+str(genome\_size)+" nucleotides ("+str(round(genome\_size\_CDS\*100/genome\_size,2))+"%)")

 output\_cds\_file.close()

 output\_xml\_file.close()

#This function extracts from a string the start and end position of a gene, print the data on screen and on cds file and register the position of all nucleotides on cds\_vector

#by changing the '0' value to 1. The change only occurs once for a nucleotide.

def write\_start\_end\_gene(range\_gene, output\_cds\_file,cds\_vector):

 indexRange=range\_gene.find("..")

 print("\t\t"+range\_gene[:indexRange]+"\t"+range\_gene[indexRange+2:], end = '')

 output\_cds\_file.write(range\_gene[:indexRange]+";"+range\_gene[indexRange+2:])

 for i in range(int(range\_gene[:indexRange]),int(range\_gene[indexRange+2:])+1):

 if (cds\_vector[i]==0):

 cds\_vector[i]=cds\_vector[i]+1

#Function to check if files are OK

def checkInputFiles():

 #Check if all the necessary files names are passed as arguments

 if (len(sys.argv)!=2 or sys.argv[1].find(".txt")==-1):

 print ("\nUsage:\npython getCDSGenBank.py [file\_path\_name.txt]")

 sys.exit(0)

 fileName\_txt=sys.argv[1]

 #Check if path/files exists

 if (not (path.exists(fileName\_txt))):

 print("\nOne or more files not found! Check the path and file names.\n")

 exit(0)

 return fileName\_txt

def main():

 fileName\_txt=checkInputFiles()

 query\_IDs=readIDs(fileName\_txt)

 #The key words are used to read the '.xml' format return by NCBI API and extract the genes data from it

 key\_words=["<GBSeq\_locus>", "<GBSeq\_length>", "<GBSeq\_definition>", "<GBFeature\_key>gene</GBFeature\_key>", "<GBFeature\_location>","<GBQualifier\_name>","<GBQualifier\_value>"]

 #This variable store ids that returns a empty result

 error\_ids=""

 for item in query\_IDs:

 print("\n\nQueryng ID: "+item[:len(item)-6]+"\n\n")

 #Get xml with GenBAnk data from NCBI

 xml\_esearch=getXMLNCBI(item)

 if (xml\_esearch.find("<ERROR>Empty result - nothing to do</ERROR>")==-1):

 xml\_esearch=xml\_esearch.splitlines()

 generateXMLCDS(item,xml\_esearch,key\_words)

 else:

 error\_ids=error\_ids+item[:len(item)-6]+"\n"

 print("\n--------------------------------------------------------------------------------------------\n")

 #If some ID returned empty, list them

 if (len(error\_ids)>0):

 print("\n--------------------------------------------------------------------------------------------\n")

 print("\nThe following IDs returned a empty result:\n"+error\_ids+"\nCheck these IDs and try again\n")

 print("\n--------------------------------------------------------------------------------------------\n")

if \_\_name\_\_ == '\_\_main\_\_':

 main()

 #This script is part of supplementary documents of "Impact of Introns and Homing Endonucleases on Structural Mitogenome Shaping in Hypocreales"

#submitted to Frontiers in Microbiology, section Fungi and Their Interactions

#Manuscript ID: 531057

#Authors: Paula Fonseca, Fernanda Badotti, Ruth De-Paula, Daniel Araújo, Dener Eduardo Bortolini, Luiz-Eduardo Del-Bem, Vasco Ariston De Carvalho Azevedo,

#Bertram Brenig, Eric Roberto Guimarães Rocha Aguiar, Aristóteles Góes-Neto

#This script uses a gff and fasta files of a target species to get the sequences of genes of interest (GOI)

#The gff file provide the start and end positions of each GOI.

#The output file is in fasta format, with the names (preceded by '>') and the sequence of genes of interest (GOI)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Run the code in Python 3+ #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

import sys

import os.path

from os import path

def checkInputFiles():

 #Check if all the necessary files names are passed as arguments

 if (len(sys.argv)!=3 or sys.argv[1].find(".gff")==-1 or sys.argv[2].find(".fasta")==-1):

 print ("\nUsage:\npython getGeneSeqGff.py [file\_path\_name.gff] [file\_path\_name.fasta]\n\n")

 sys.exit(0)

 gff\_file\_name=sys.argv[1]

 fasta\_file\_name=sys.argv[2]

 #Check if path/files exists

 if (not (path.exists(gff\_file\_name) or path.exists(fasta\_file\_name))):

 print("\nOne or more files not found! Check the path and file names.\n")

 exit(0)

 return gff\_file\_name, fasta\_file\_name

#Reads whole sequence from input fasta file

def readFasta(fasta\_file):

 #Position 0 of whole\_genome will not be used

 whole\_genome=" "

 for line in fasta\_file:

 if (line.find(">")==-1):

 whole\_genome=whole\_genome+line.strip()

 fasta\_file.close()

 return whole\_genome

#This function check if the gene\_name is present in the whitelist array, case true, it saves the name and sequence of the gene

def search\_genes(gene\_name,gene\_sequence, output\_file, GOI):

 for GOI in GOI:

 if (gene\_name.startswith(GOI)):

 output\_file.write(">"+gene\_name+"\n"+gene\_sequence+"\n\n")

 break

#Read the gff file to extract data.

#The gff file contains 1 gene per row with several values ordered by 'tab'. Its straight forward to get the name and positions of a single gene

#and retrieve th sequence from the whole\_genome

def readGffSelGenes(GOI, gff\_file, output\_file, whole\_genome):

 for line in gff\_file:

 values=line.split("\t")

 #Start position is at index 3

 start\_gene\_position=values[3]

 #End position is at index 4

 end\_gene\_position=values[4]

 #Name is at index 8

 gene\_name=values[8][values[8].find("Name=")+5:].strip()

 #Get gene sequence

 gene\_sequence=whole\_genome[int(start\_gene\_position):int(end\_gene\_position)+1]

 #Verify if it is on gene\_whitelist to save in output file

 search\_genes(gene\_name,gene\_sequence,output\_file,GOI)

def main():

 GOI={"rrnL","rps3","nad2","nad3","atp9","cox2","nad4l","nad5","cob","cox1","nad1","nad4","atp8","atp6","rrnS","cox3","nad6"}

 gff\_file\_name, fasta\_file\_name=checkInputFiles()

 gff\_file=open(gff\_file\_name,'r')

 fasta\_file=open(fasta\_file\_name,'r')

 whole\_genome=readFasta(fasta\_file)

 #Get ID specie from gff file name

 #The strip will remove '.\' that appear on console in Windows 10 before path\filename

 if (os.name=="nt"):

 gff\_file\_name=gff\_file\_name.strip(".\\")

 output\_file\_name=gff\_file\_name[0:gff\_file\_name.find(".")]+"\_GOI.fasta"

 #Open output file with '\_GOI.fasta' extension

 output\_file=open(output\_file\_name,'w')

 readGffSelGenes(GOI,gff\_file,output\_file, whole\_genome)

 gff\_file.close()

 output\_file.close()

 print("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

 print("\nResults saved in: "+output\_file\_name)

 print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n")

if \_\_name\_\_ == '\_\_main\_\_':

 main()

 #This script is part of supplementary documents of "Impact of Introns and Homing Endonucleases on Structural Mitogenome Shaping in Hypocreales"

#submitted to Frontiers in Microbiology, section Fungi and Their Interactions

#Manuscript ID: 531057

#Authors: Paula Fonseca, Fernanda Badotti, Ruth De-Paula, Daniel Araújo, Dener Eduardo Bortolini, Luiz-Eduardo Del-Bem, Vasco Ariston De Carvalho Azevedo,

#Bertram Brenig, Eric Roberto Guimarães Rocha Aguiar, Aristóteles Góes-Neto

#This script uses the output file of Mfannot to list and save uORFs of a target species

#As described in https://github.com/BFL-lab/Mfannot, Mfannot is a program for annotation of mitochondrial and plastid genomes

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Run the code in Python 3+ #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# -\*- Coding: UTF-8 -\*-

#coding: utf-8

import sys

import os.path

from os import path

#Function to check if files are OK

def checkMfannotFile():

 #Check if all the necessary files names are passed as arguments

 if (len(sys.argv)!=2):

 print ("\nUsage:\npython getuORFs.py [file\_path\_name]")

 sys.exit(0)

 mfannot\_file\_name=sys.argv[1]

 #Check if path/files exists

 if (not (path.exists(mfannot\_file\_name))):

 print("\nOne or more files not found! Check the path and file names.\n")

 exit(0)

 #Open input file

 input\_file=open(mfannot\_file\_name,'r')

 #Check if input file is a Mfannot

 if (input\_file.readline().find("mfannot")==-1):

 print("\nThe file is empty or is not a mfannot output file\n")

 input\_file.close()

 exit(0)

 #Get ID specie from Mfannot file name

 #The strip will remove '.\' that appear on console in Windows 10 before path\filename

 if (os.name=="nt"):

 mfannot\_file\_name=mfannot\_file\_name.strip(".\\")

 output\_file\_name=mfannot\_file\_name[0:mfannot\_file\_name.find(".")]

 print(mfannot\_file\_name)

 print(output\_file\_name)

 #Open uORFs output file

 output\_file=open(output\_file\_name+".uORFs",'w')

 return input\_file, output\_file

#This function look if a string contain a uORfs\_name and store it in uORfs\_name\_vector

def findOrfs(str\_name, uORfs\_name\_vector):

 if (str\_name.find("orf")!=-1):

 uORfs\_name\_vector.append(str\_name)

def getuORFSNamesMfannot(uORfs\_name\_vector, input\_file):

 #find\_str\_gene is a boolean variable that signalize where the block of gene names start and end

 #The Mfannot file lists all gene names in a tab format starting at line 4

 find\_str\_gene=0

 #Loop to read the Mfannot input file

 for line in input\_file:

 #If the string "List of genes added", then the boolean find\_str\_gene receives 1, signalizing that we are reading the block with gene names

 if (line.find("List of genes added")!=-1):

 find\_str\_gene=1

 #If find\_str\_gene is true, then we read gene names

 if (find\_str\_gene==1):

 #Check if it is at end of gene names block

 if (line.find("end mfannot")!=-1):

 find\_str\_gene=0 #para terminar de ler até o end do mfannot.

 #The gene names are structured in 3 columns of regular spaced sizes, so we read each one and stores in uORfs\_name\_vector

 else:

 findOrfs(line[8:29].rstrip(' '),uORfs\_name\_vector)

 findOrfs(line[29:50].rstrip(' '),uORfs\_name\_vector)

 findOrfs(line[50:70].rstrip(' '),uORfs\_name\_vector)

def getuORFsStartEndSeq(uORfs\_name\_vector, input\_file,output\_file):

 for uORfs\_name in uORfs\_name\_vector:

 #This sets the position of reading the input\_file at the start

 input\_file.seek(0)

 #This boolean tell us when a sequence of a specific uORF begins

 bool\_start\_seq=0

 #orf\_seq store the orf's sequence

 orf\_seq=""

 orf\_start\_position=""

 orf\_end\_position=""

 detailed\_orf\_name=""

 #num\_index store the index +1 after the number position in sequences lines

 num\_index=-1

 #Loop to read the input\_file

 for line in input\_file:

 #When we find the start line with the uORFS\_name, bool\_start\_seq receives 1 (true)

 if (line.find("-" + uORfs\_name)!=-1 and line.find(" ==> start")!=-1):

 detailed\_orf\_name=line[1:line.find(" ==> start")].strip()

 bool\_start\_seq=1

 else:

 if (bool\_start\_seq==1):

 #num\_index store the index +1 after the number position in sequences lines

 num\_index=line.find(" ",2)

 #If orf\_start\_position is empty and bool\_start\_seq==1, then we get the start position of the orf

 if (orf\_start\_position==""):

 orf\_start\_position=line[:num\_index].strip()

 #Check if its the end of the sequence of uORfs\_name, then break case true

 if (line.find("-" + uORfs\_name)!=-1 and line.find(" ==> end")!=-1):

 print(">"+detailed\_orf\_name)

 print("+"+orf\_start\_position)

 print("-"+str(orf\_end\_position))

 print("@"+orf\_seq)

 output\_file.write(">"+detailed\_orf\_name+"\n")

 output\_file.write("+"+orf\_start\_position+"\n")

 output\_file.write("-"+str(orf\_end\_position)+"\n")

 output\_file.write("@"+orf\_seq+"\n\n")

 #Here we reset variables and let the loop go to the end, as is possible to have another copy

 #forward in the file

 orf\_seq=""

 orf\_start\_position=""

 orf\_end\_position=""

 detailed\_orf\_name=""

 bool\_start\_seq=0

 elif(line.find(";")==-1):

 orf\_seq=orf\_seq + line[num\_index:].strip()

 #Calculate the orf\_end\_position

 orf\_end\_position=int(line[:num\_index].strip())+len(line[num\_index:].strip())-1

def main():

 input\_file,output\_file=checkMfannotFile()

 #uORfs\_name\_vector is a array that stores the name of the uORFs listed in Mfannot file

 uORfs\_name\_vector=[]

 getuORFSNamesMfannot(uORfs\_name\_vector, input\_file)

 getuORFsStartEndSeq(uORfs\_name\_vector, input\_file, output\_file)

 output\_file.close()

 input\_file.close()

if \_\_name\_\_ == '\_\_main\_\_':

 main()

 #This script is part of supplementary documents of "Impact of Introns and Homing Endonucleases on Structural Mitogenome Shaping in Hypocreales"

#submitted to Frontiers in Microbiology, section Fungi and Their Interactions

#Manuscript ID: 531057

#Authors: Paula Fonseca, Fernanda Badotti, Ruth De-Paula, Daniel Araújo, Dener Eduardo Bortolini, Luiz-Eduardo Del-Bem, Vasco Ariston De Carvalho Azevedo,

#Bertram Brenig, Eric Roberto Guimarães Rocha Aguiar, Aristóteles Góes-Neto

#This script calculates the GC content of whole genome, CDS and genes in the uORFs file

#The files used are as follow:

# -uORFs - Generated by getuORFs.py script

# -cds - Generated by getGenesGenBank.py script

# -fasta - Donwloaded from NCBI

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Run the code in Python 3+ #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# -\*- Coding: UTF-8 -\*-

#coding: utf-8

import sys

import re

import os.path

from os import path

def checkInputFiles():

 # #Check if all the necessary files names are passed as arguments

 if (len(sys.argv)!=4 or sys.argv[1].find(".uORFs")==-1 or sys.argv[2].find(".cds")==-1 or sys.argv[3].find(".fasta")==-1):

 print ("\nUsage:\npython GC\_Contet\_uORFs.py [file\_path\_name.uORFs] [file\_path\_name.cds] [file\_path\_name.fasta]\n")

 sys.exit(0)

 #Get path/file names

 uORFs\_file\_name=sys.argv[1]

 cds\_file\_name=sys.argv[2]

 fasta\_file\_name=sys.argv[3]

 #Check if path/files exists

 if (not (path.exists(uORFs\_file\_name) or path.exists(cds\_file\_name) or path.exists(fasta\_file\_name))):

 print("\nOne or more files not found! Check the path and file names.\n")

 exit(0)

 #Open input files

 uORFs\_file=open(uORFs\_file\_name,'r')

 cds\_file=open(cds\_file\_name, 'r')

 fasta\_file=open(fasta\_file\_name,'r')

 #Open output files. The ID filename in uORFs file is used to generate the result files ('.gct' and '.csv')

 if (os.name=="nt"):

 uORFs\_file\_name=uORFs\_file\_name.strip(".\\")

 output\_file\_name=uORFs\_file\_name[0:uORFs\_file\_name.find(".")]

 output\_gct\_file=open(output\_file\_name+".gct",'w')

 #The csv file was generated to help analyze the results. Each row of 'csv' file represent a nucleotide position in the whole genome.

 #The idea is as follows:

 #Row value= 0 = indicates the nucleotide belongs a non coding region

 #Row value= 1 = indicates the nucleotide belongs a coding region

 #Row value= 2 = indicates the nucleotide belongs a coding region and to 2 genes.

 #Row value= 10 = indicates the nucleotide belongs a non coding region and to a uORF

 #Row value= 11 = indicates the nucleotide belongs a coding region and to a uORF

 #Row value= 12 = indicates the nucleotide belongs a coding region, to a uORF and 2 genes

 #Row value= 22 = indicates the nucleotide belongs a coding region, to 2 uORFs and 2 genes

 #and so on

 #The strip will remove '.\' that appear on console in Windows 10 before path\filename

 output\_csv\_file=open(output\_file\_name+".csv",'w')

 return uORFs\_file,cds\_file,fasta\_file,output\_gct\_file,output\_csv\_file

#Based on cds file, this function creates a numerical array (genome\_array) that represents where the coding, non coding and uORFs are, returning it

def createGenomeArray(cds\_file):

 #genome\_array represent the whole genome. Position 0 is not used.

 genome\_array=[]

 genome\_size=0

 #Loop to get data from cds file

 for line in cds\_file:

 #get total genome size from cds file

 if (line.rfind("Genome size: ")!=-1):

 genome\_size=int(line[13:])

 #Instantiate size of genome in genome\_array and populates with value=0

 genome\_array=[0]\*(genome\_size+1)

 #Get start and end positions of coding regions (genes on cds)

 if (line.find(";")!=-1):

 aux\_index=line.find(";")

 line=line.strip()

 start=int(line[:aux\_index])

 end=int(line[aux\_index+1:line.find("#")])

 #Loop to register nucleotides that belong to coding regions, based on start and end positions retrieved

 #This adds +1 every time a nucleotide belong to a gene

 for i in range(start,end+1):

 genome\_array[i]=genome\_array[i]+1

 return genome\_array

def checkCG(nc\_char):

 if (nc\_char=='C' or nc\_char=='G'):

 return True

 else:

 return False

#This function calculates the GC content of the coding and non coding regions of a sequence. Using the genome\_array as input,

#its possible to determinte the GC content in coding and non coding region. As well check results in csv file about them and the instersection

# with the uORFs

def gcContentCalc(start, end, sequence, genome\_array):

 #seq\_cds store the sequence of nucleotides that are part of the coding region. Those nucleotides that are not part of the coding region are replaced by '-'

 seq\_cds=""

 #sum\_gc\_nuc store the sum of GC nucleotides in the sequence

 sum\_gc\_nuc=0

 #sum\_gc\_nuc\_cds store the sum of GC nucleotides that are part of coding region in the sequence

 sum\_gc\_nuc\_cds=0

 #nuc\_cds store the sum of ALL nucleotides that are part of coding region in the sequence

 nuc\_cds=0

 #For every nucleotide in the sequence

 for i in range(start,end+1):

 #Check if its part of a coding region in genome\_array

 #Because uORFs nucleotides adds +10 to genome\_array and is possible that they are not part of coding regions,

 #we get the remainder of division by 10

 if (genome\_array[i]%10>0):

 #if it is G or C

 if (checkCG(sequence[i-start])):

 #Add 1 to sum sum\_gc\_nuc\_cds

 sum\_gc\_nuc\_cds=sum\_gc\_nuc\_cds+1

 #Add nucleotide to seq\_cds

 seq\_cds=seq\_cds+sequence[i-start]

 #And 1 to nuc\_cds

 nuc\_cds=nuc\_cds+1

 #if not part of coding region, '-' replace the nucleotide in seq\_cds

 else:

 seq\_cds=seq\_cds+'-'

 #Indenpedent of being part of coding region

 #If C or G

 if (checkCG(sequence[i-start])):

 #Add 1 to sum\_gc\_nuc

 sum\_gc\_nuc=sum\_gc\_nuc+1

 #Adding +10 to genome\_array will help later check where are the nucleotides that belong to uORFs in csv file

 #Values greater than or equal 10

 genome\_array[i]=genome\_array[i]+10

 #GC\_nc\_ratio\_cds show the proportion of GC nucleotides in the coding region of the sequence

 GC\_nc\_ratio\_cds=0

 if (nuc\_cds>0):

 GC\_nc\_ratio\_cds=sum\_gc\_nuc\_cds/nuc\_cds\*100

 #The next command line returns: proportion of GC nucleotides in the sequence

 #Nucleotides in coding region of the sequence

 #Total of GC nucleotides in the sequence

 #Total of GC nucleotides in the coding region of the sequence

 #Total of nucleotides in the coding region of the sequence

 #Size of sequence

 #and the proportion of GC nucleotides in the coding region of the sequence

 return sum\_gc\_nuc/(end+1-start)\*100, seq\_cds, sum\_gc\_nuc, sum\_gc\_nuc\_cds,nuc\_cds,end+1-start, GC\_nc\_ratio\_cds

#This function read the whole genome from fasta file

def readWholeGenome(fasta\_file):

 #The position 0 of whole\_genome will not be used

 whole\_genome=" "

 for line in fasta\_file:

 if(line[0]!=">"):

 line=line.upper()

 whole\_genome=whole\_genome+line.strip()

 fasta\_file.close()

 return whole\_genome

#Function that calculate uORFs GC content in coding and non coding regions

def uORFsFileGCCalc(uORFs\_file, genome\_array, output\_gct\_file):

 name\_orf=""

 #Total of GC nucleotides in ORFs

 gc\_total\_orfs=0

 #Total of GC nucleotides in ORFs that are part of coding regions

 gc\_total\_orfs\_cds=0

 #Total size in nucleotides of the ORFs

 sum\_size\_uorfs=0

 #Total size in nucleotides of the ORFs in coding regions

 sum\_size\_uorfs\_cds=0

 for line in uORFs\_file:

 if (line.find(">")!=-1):

 name\_orf=line[1:]

 elif (line.find("+")!=-1):

 start\_orf=int(line[1:])

 elif (line.find("-")!=-1):

 end\_orf=int(line[1:])

 elif (line.find("@")!=-1):

 seq\_orf=line[1:].upper()

 print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

 print(name\_orf)

 #call function that calculate GC Content and update genome\_array

 ratio\_GC\_orf,seq\_in\_cds, total\_GC\_nc\_orf, total\_GC\_nc\_orf\_cds, total\_nc\_orf\_cds,size\_orf,ratio\_GC\_orf\_cds=gcContentCalc(start\_orf, end\_orf, seq\_orf,genome\_array)

 print("uORf original sequence:\n"+seq\_orf+"\nuORF sequence in CDS:\n"+seq\_in\_cds)

 print(start\_orf, end\_orf)

 print("GC Content of Orf:",round(ratio\_GC\_orf,2))

 print("GC Content of Orf in CDS:",round(ratio\_GC\_orf\_cds,2))

 output\_gct\_file.write(name\_orf)

 output\_gct\_file.write(str(start\_orf)+","+str(end\_orf)+"\n")

 output\_gct\_file.write("uORf original sequence:\n"+seq\_orf.rstrip()+"\nuORF sequence in CDS:\n"+seq\_in\_cds+"\n")

 output\_gct\_file.write("Conteudo GC Orf: "+str(round(ratio\_GC\_orf,2))+"\nConteudo GC Orf CDS: "+str(round(ratio\_GC\_orf\_cds,2))+"\n\n")

 gc\_total\_orfs=gc\_total\_orfs + total\_GC\_nc\_orf

 gc\_total\_orfs\_cds=gc\_total\_orfs\_cds + total\_GC\_nc\_orf\_cds

 sum\_size\_uorfs=sum\_size\_uorfs+size\_orf

 sum\_size\_uorfs\_cds=sum\_size\_uorfs\_cds+total\_nc\_orf\_cds

 return gc\_total\_orfs,gc\_total\_orfs\_cds,sum\_size\_uorfs,sum\_size\_uorfs\_cds

#Calculate GC content in whole Genome

def wholeGenomeGCCalc(output\_csv\_file,output\_gct\_file,whole\_genome, genome\_array, gc\_total\_orfs, gc\_total\_orfs\_cds, sum\_size\_uorfs, sum\_size\_uorfs\_cds):

 #Total of nucleotides in the whole genome that belongs to coding regions

 sum\_nc\_genome\_cds=0

 #Total of nucleotides in the whole genome that belongs to non coding regions

 sum\_nc\_genome\_noncod=0

 #Total of GC nucleotides in the whole genome that belongs to coding regions

 sum\_GC\_nc\_cds=0

 #Total of GC nucleotides in the whole genome that belongs to coding regions

 sum\_GC\_nc\_noncod=0

 genome\_size=len(genome\_array)-1

 for i in range(1,len(genome\_array)):

 output\_csv\_file.write(str(genome\_array[i])+"\n")

 #Check if nucleotide is part of coding region

 #Because uORFs nucleotides adds +10 to genome\_array and is possible that they are not part of coding regions,

 #we get the remainder of division by 10

 if (genome\_array[i]%10>0):

 if (checkCG(whole\_genome[i])):

 sum\_GC\_nc\_cds= sum\_GC\_nc\_cds+1

 sum\_nc\_genome\_cds=sum\_nc\_genome\_cds+1

 else:

 if (checkCG(whole\_genome[i])):

 sum\_GC\_nc\_noncod= sum\_GC\_nc\_noncod+1

 sum\_nc\_genome\_noncod=sum\_nc\_genome\_noncod+1

 print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

 print("\n")

 print("------------------------------------------------------------------------------------------------------------------------------------")

 print("Whole genome total size = "+str(genome\_size)+" nucleotides, where "+ str(sum\_nc\_genome\_cds)+" nucleotides ("+str(round(sum\_nc\_genome\_cds/genome\_size\*100,2)) \

 +"%) belongs to coding regions (CDS) and "+ str(sum\_nc\_genome\_noncod)+" nucleotides ("+str(round(sum\_nc\_genome\_noncod/genome\_size\*100,2))+"%) belongs to non coding regions (NC)")

 print("Whole genome GC content = "+str(sum\_GC\_nc\_cds+sum\_GC\_nc\_noncod)+" of "+str(genome\_size)+" nucleotides ("+str(round((sum\_GC\_nc\_cds+sum\_GC\_nc\_noncod)/genome\_size\*100,2))+"%)")

 print("GC content in coding regions = "+str(sum\_GC\_nc\_cds)+" of "+str(sum\_nc\_genome\_cds)+" nucleotides ("+str(round(sum\_GC\_nc\_cds/sum\_nc\_genome\_cds\*100,2))+"%)")

 print("GC content in non coding regions = "+str(sum\_GC\_nc\_noncod)+" of "+str(sum\_nc\_genome\_noncod)+" nucleotides (" +str(round(sum\_GC\_nc\_noncod/sum\_nc\_genome\_noncod\*100,2))+"%)")

 print("uORFs total size = "+ str(sum\_size\_uorfs) + " nucleotides, corresponding to " + str(round(sum\_size\_uorfs/genome\_size\*100,2))+ "% "+"of whole genome")

 print("uORfs GC content = " +str(gc\_total\_orfs)+" of "+str(sum\_size\_uorfs)+" nucleotides ("+str(round(gc\_total\_orfs/sum\_size\_uorfs\*100,2))+"%)")

 print("uORFs total size in coding regions (CDS) = "+ str(sum\_size\_uorfs\_cds) + " nucleotides")

 print("uORFs total size in non coding regions (NC) = "+ str(sum\_size\_uorfs-sum\_size\_uorfs\_cds) + " nucleotides")

 print("uORFs GC content in coding regions (CDS) = " +str(gc\_total\_orfs\_cds)+" of "+str(sum\_size\_uorfs\_cds) + " nucleotides ("+ str(round(gc\_total\_orfs\_cds/sum\_size\_uorfs\_cds\*100,2))+"%)")

 if (sum\_size\_uorfs-sum\_size\_uorfs\_cds!=0):

 print("uORFs GC content in non coding regions (NC) = " +str(gc\_total\_orfs-gc\_total\_orfs\_cds)+" of "+str(sum\_size\_uorfs-sum\_size\_uorfs\_cds) + " nucleotides ("+ \

 str(round((gc\_total\_orfs-gc\_total\_orfs\_cds)/(sum\_size\_uorfs-sum\_size\_uorfs\_cds)\*100,2))+"%)")

 print("------------------------------------------------------------------------------------------------------------------------------------")

 output\_gct\_file.write("\n")

 output\_gct\_file.write("------------------------------------------------------------------------------------------------------------------------------------\n")

 output\_gct\_file.write("Whole genome total size = "+str(genome\_size)+" nucleotides, where "+ str(sum\_nc\_genome\_cds)+" nucleotides ("+str(round(sum\_nc\_genome\_cds/genome\_size\*100,2)) \

 +"%) belongs to coding regions (CDS) and "+ str(sum\_nc\_genome\_noncod)+" nucleotides ("+str(round(sum\_nc\_genome\_noncod/genome\_size\*100,2))+"%) belongs to non coding regions (NC)\n")

 output\_gct\_file.write("Whole genome GC content = "+str(sum\_GC\_nc\_cds+sum\_GC\_nc\_noncod)+" of "+str(genome\_size)+" nucleotides ("+str(round((sum\_GC\_nc\_cds+sum\_GC\_nc\_noncod)/genome\_size\*100,2))+"%)\n")

 output\_gct\_file.write("GC content in coding regions = "+str(sum\_GC\_nc\_cds)+" of "+str(sum\_nc\_genome\_cds)+" nucleotides ("+str(round(sum\_GC\_nc\_cds/sum\_nc\_genome\_cds\*100,2))+"%)\n")

 output\_gct\_file.write("GC content in non coding regions = "+str(sum\_GC\_nc\_noncod)+" of "+str(sum\_nc\_genome\_noncod)+" nucleotides (" +str(round(sum\_GC\_nc\_noncod/sum\_nc\_genome\_noncod\*100,2))+"%)\n")

 output\_gct\_file.write("uORFs total size = "+ str(sum\_size\_uorfs) + " nucleotides, corresponding to " + str(round(sum\_size\_uorfs/genome\_size\*100,2))+ "% "+"of whole genome\n")

 output\_gct\_file.write("uORfs GC content = " +str(gc\_total\_orfs)+" of "+str(sum\_size\_uorfs)+" nucleotides ("+str(round(gc\_total\_orfs/sum\_size\_uorfs\*100,2))+"%)\n")

 output\_gct\_file.write("uORFs total size in coding regions (CDS) = "+ str(sum\_size\_uorfs\_cds) + " nucleotides\n")

 output\_gct\_file.write("uORFs total size in non coding regions (NC) = "+ str(sum\_size\_uorfs-sum\_size\_uorfs\_cds) + " nucleotides\n")

 output\_gct\_file.write("uORFs GC content in coding regions (CDS) = " +str(gc\_total\_orfs\_cds)+" of "+str(sum\_size\_uorfs\_cds) + " nucleotides ("+ str(round(gc\_total\_orfs\_cds/sum\_size\_uorfs\_cds\*100,2))+"%)\n")

 if (sum\_size\_uorfs-sum\_size\_uorfs\_cds!=0):

 output\_gct\_file.write("uORFs GC content in non coding regions (NC) = " +str(gc\_total\_orfs-gc\_total\_orfs\_cds)+" of "+str(sum\_size\_uorfs-sum\_size\_uorfs\_cds) + " nucleotides ("+ \

 str(round((gc\_total\_orfs-gc\_total\_orfs\_cds)/(sum\_size\_uorfs-sum\_size\_uorfs\_cds)\*100,2))+"%)\n")

 output\_gct\_file.write("------------------------------------------------------------------------------------------------------------------------------------\n")

def main():

 uORFs\_file,cds\_file,fasta\_file,output\_gct\_file,output\_csv\_file =checkInputFiles()

 #Call function that reads data from 'cds' file, creating genome\_array

 genome\_array = createGenomeArray(cds\_file)

 #Call function to read whole genome from fasta file

 whole\_genome=readWholeGenome(fasta\_file)

 #Call function to calculate GC Content of uORfs

 gc\_total\_orfs, gc\_total\_orfs\_cds, sum\_size\_uorfs, sum\_size\_uorfs\_cds=uORFsFileGCCalc(uORFs\_file,genome\_array,output\_gct\_file)

 #Call function to calculate GC Content of whole genome

 wholeGenomeGCCalc(output\_csv\_file,output\_gct\_file, whole\_genome, genome\_array, gc\_total\_orfs, gc\_total\_orfs\_cds, sum\_size\_uorfs, sum\_size\_uorfs\_cds)

 print("\n\nResults saved on: "+str(output\_gct\_file.name)+" e "+str(output\_csv\_file.name)+"\n")

 uORFs\_file.close()

 cds\_file.close()

 output\_csv\_file.close()

 output\_gct\_file.close()

if \_\_name\_\_ == '\_\_main\_\_':

 main()