README.md

Orthologous clustering of secretomes of Parabasalids & Anaeramoebids

Author: Virág Varga

Date: 20.09.2021

Description

A `conda` environment was created for this project (completed on the LU Bioinformatics server):

```
#create environment
conda create -n trich_parab
#activate environment
conda activate trich_parab
###
#the pandas python module was not automatically installed
#it can easily be installed via anaconda
conda install pandas
```

SignalP

SignalP is a signal peptide detection software that predicts protein targeting for secretion. The program homepage is available here: http://www.cbs.dtu.dk/services/SignalP/

The executable file download link is available for academic use from here: https://services.healthtech.dtu.dk/software.php

A license is necessary in order to install the program, but it is free for academic use.

Installed on the server at: /usr/local/bin/signalp

eggNOG

The eggNOG database is used for orthology prediction and functional annotation. The website is available here: http://eggnog5.embl.de/#/app/home

Download information for emapper, which is the associated tool used for annotation of a collection of proteins, is available here: http://eggnog5.embl.de/#/app/downloads

Installed on the server at: /usr/local/bin/emapper.py

DeepLoc

Not on the server, so needs to be installed:

Files obtained from DeepLoc website: https://services.healthtech.dtu.dk/service.php?DeepLoc-1.0

A license is necessary in order to install the program, but it is free for academic use.

Tool can be used online at: http://www.cbs.dtu.dk/services/DeepLoc/

```
#place files in the bin/ directory with trich_parab environment activated
#unpack tar file
tar -xvf deeploc-1.0.All.tar.gz
#installing dependencies
#it turns out some of these dependencies are quite old
#so need an environment with an older version of python
conda create -n DeepLoc_usage python=3.5.1
conda activate DeepLoc_usage
#Lasagne==0.2.dev1
pip install -r https://raw.githubusercontent.com/Lasagne/Lasagne/master/requirements.txt
pip install https://github.com/Lasagne/Lasagne/archive/master.zip
#installing theano generally installs the other requirements, too:
#Numpy
conda install -c anaconda numpy
#Scipy
conda install -c anaconda scipy
#Theano==1.0.1
conda install -c conda-forge theano=1.0.1
#six==1.11.0
conda install -c conda-forge six=1.11.0
#also need to install the following:
#mkl-service
conda install mkl-service
#the above isn't mentioned in the installation instructions,
#but an error code informed me it needed to be installed
#add deeploc to the .bashrc file, like so:
#export PATH="$PATH:/home/inf-47-2020/bin/DeepLoc/deeploc-1.0/bin"
#and then source it:
source ~/.bashrc
```

OrthoFinder

OrthoFinder can be installed from the source on GitHub (https://github.com/davidemms/OrthoFinder), or via `conda`:

```
#in the trich_parab conda environment
conda install orthofinder
conda update orthofinder
```

COUNT

From the website (here: http://www.iro.umontreal.ca/~csuros/gene_content/count.html): Count is a software package for the evolutionary analysis of homolog family sizes (phylogenetic profiles), or other numerical census-type characters along a phylogeny.

Note that this program requires Java to be installed in order for it to run.

```
#program installation
#following directions on the program website: http://www.iro.umontreal.ca/~csuros/gene_content/count.html
#placed the .tgz file into the bin/ directory on the server
tar -zxvf Count.tgz
#need to add the new ~/bin/Count/ directory to the .bashrc file
#then can use program like so:
java -jar Count.jar
###
#the program runs on Java, and has an interactive window
#which can be accessed remotely using Xming & PuTTY to do X-11 forwarding
#however, it is much simpler to install and use the program from my local Windows computer
#in that case, I merely unpacked the download Count.zip file
#this created a folder from which the program can be opened by clicking on an icon
```

Python & Spyder

The Spyder IDE was used on a local Windows computer for testing Python scripts.

Two non-standard Python modules were utilized in this research process.

The numpy library aids in manipulation of arrays. More information can be found at their website, here: https://numpy.org/

The pandas library facilitates the manipulation of dataframes in Python. More information is available on their website, here: https://pandas.pydata.org/

Both libraries can be installed with `conda`, like so:

```
#install pandas
conda install -c anaconda pandas
#install numpy
conda install -c anaconda numpy
```

R & RStudio

The RStudio IDE was used on a local Windows computer for analysis of data.

The UpSetR package was used to create an UpSet plot. The package is distributed by CRAN; more information is available here: https://cran.r-project.org/web/packages/UpSetR/index.html

It can be installed as follows:

```
#package installation
install.packages("upsetr")
```

FigTree

This is a phylogenetic tree visualization and analysis software. The program code can be downloaded from here: https://github.com/rambaut/figtree/releases

Note: FigTree requires at least Java 1.5 to run. Java can be downloaded from the website, here: https://www.oracle.com/java/technologies/javase-downloads.html

```
#in the bin/ in the fistulina environment on the server
tar -zxvf FigTree_v1.4.4.tgz
#adding the program to the PATH:
#export PATH="$PATH:/home/inf-47-2020/bin/FigTree_v1.4.4/bin"
#source the .bashrc
source ~/.bashrc
###
#for installation on Windows 10:
#download the FigTree_v1.4.4.zip file from the GitHub site above
#extract the files from the archive
#copy the FigTree_v1.4.4/ directory to a desired Program storage directory
#run the application by double-clicking the application icon
#which is in the unzipped folder
```

Inkscape

Inkscape is a vector graphics editor that was used in order to edit SVG output files into the finalized figures that were used in the report. The GUI can be downloaded from here: https://inkscape.org/release/inkscape-1.1/

draw.io/diagrams.net

The draw.io software is a freely available software usable for creating simple diagrams. Both an online-only and desktop version of the tool are available from the website, here: https://drawio-app.com/

Data Collection

Species datasets were pulled either from a collection in Courtney's Drive (primarily assembled by her) and from the NCBI.

NCBI Exploration:

- Search from the Parabasalia page of the NCBI Taxonomy Browser: https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?mode=Undef&id=5719&lvl=3&lin=f&keep=1&srchmode=1&unlock
- Search for species with "Bio Project" and "Bio Sample"

NCBI Data Collection Process:

- Search URL: https://www.ncbi.nlm.nih.gov/bioproject/?term=txid5719[Organism:exp]
- Manually checked the results of the above search in order to find protein assemblies
- If the files weren't in the NCBI database, the articles associated with these entries were tracked down where possible

In this manner, annotated genomes for the following species/strains were collected:

- Trichomonas vaginalis RefSeq version
- Trichomonas vaginalis GenBank version
- Tritrichomonas foetus (Note that as a result of an error early in the pipeline, this strain is referred to as Trichomonas foetus in the rest of the README)

The 3 anaeromoebid datasets were provided by Jon Jerlström-Hultqvist. These genomes were annotated through his own research. The 3 datasetts are:

- SC strain
- BS strain
- BM strain

Additional datasets were provided by Courtney Stairs, who had previously created assemblies from publicly available RNASeq data on the NCBI. These datasets are:

- Dientamoeba fragilis
- Histomonas meleagridis
- Pentatrichomonas hominis
- Tetratrichomonas gallinarum

This constitutes 10 total assemblies of 9 species/strains.

Retrieval of Datasets

FASTA Files were retrieved to the server:

```
#NCBI datasets retrieved with `wget`
#Trichomonas vaginalis G3 GCF_000002825.2_ASM282v1
wget \
  https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/002/825/GCF_000002825.2_ASM282v1/GCF_000002825.2_ASM282v1_protein.\
  faa.gz
#Trichomonas vaginalis G3 GCA_000002825.1
wget \
  https://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/002/825/GCA_000002825.1_ASM282v1/GCA_000002825.1_ASM282v1_protein.\
  faa.gz
#Trichomonas foetus GCA_001839685.1
wget \
  https://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/001/839/685/GCA_001839685.1_ASM183968v1/GCA_001839685.1_ASM183968v1_\
  protein.faa.gz
#unpack .gz files
gunzip *.gz
```

File Exploration & Editing

The amino acid FASTA files needed to be set up in the same format, in order to streamline the analysis.

```
#examining FASTA file formats
#to check header formatting, use:
head
#to count number of sequences, use:
grep -c ">"
#to count number of lines in file, use:
wc -l
#together the above 2 can be used to determine whether the FASTA
#is 1-line or multi-line format
#it looks like Courtney's are 1-line, and the NCBI are multi-line
#Courtney's files end in *.aa.fasta
grep -c ">" *.fasta
#Dientamoeba_fragilis.43352.aa.fasta:6558
#Histomonas_meleagridis.135588.aa.fasta:6982
#Pentatrichomonas_hominis.5728.aa.fasta:44819
#Tetratrichomonas_gallinarum.5730.aa.fasta:67510
wc -l *.fasta
#   13116 Dientamoeba_fragilis.43352.aa.fasta
#   13964 Histomonas_meleagridis.135588.aa.fasta
#   89638 Pentatrichomonas_hominis.5728.aa.fasta
#  135020 Tetratrichomonas_gallinarum.5730.aa.fasta
#  251738 total
#NCBI files end in *.faa
grep -c ">" *.faa
#GCA_000002825.1_ASM282v1_protein.faa:59681
#GCA_001839685.1_ASM183968v1_protein.faa:25030
#GCF_000002825.2_ASM282v1_protein.faa:59679
wc -l *.faa
#  315386 GCA_000002825.1_ASM282v1_protein.faa
#  202344 GCA_001839685.1_ASM183968v1_protein.faa
#  315382 GCF_000002825.2_ASM282v1_protein.faa
#  833112 total
#I don't like multi-line, so I'll convert them to 1-line FASTAs
#not possible to overwrite the files, so saving to new files
awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}  END {printf("\n");}' \
  < GCA_000002825.1_ASM282v1_protein.faa \
  > Trichomonas_vaginalis_GenBank_start.PRJNA16084.fasta
awk '{if (NR==1 && NF==0) next};1' Trichomonas_vaginalis_GenBank_start.PRJNA16084.fasta \
  > Trichomonas_vaginalis_GenBank.PRJNA16084.fasta
awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}  END {printf("\n");}' \
  < GCF_000002825.2_ASM282v1_protein.faa \
  > Trichomonas_vaginalis_RefSeq_start.G3.fasta
awk '{if (NR==1 && NF==0) next};1' Trichomonas_vaginalis_RefSeq_start.G3.fasta \
  > Trichomonas_vaginalis_RefSeq.G3.fasta
awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}  END {printf("\n");}' \
  < GCA_001839685.1_ASM183968v1_protein.faa \
  > Trichomonas_foetus_start.PRJNA345179.fasta
awk '{if (NR==1 && NF==0) next};1' Trichomonas_foetus_start.PRJNA345179.fasta \
  > Trichomonas_foetus.PRJNA345179.fasta
#final checks
grep -c ">" *.fasta
#Dientamoeba_fragilis.43352.aa.fasta:6558
#Histomonas_meleagridis.135588.aa.fasta:6982
#Pentatrichomonas_hominis.5728.aa.fasta:44819
#Tetratrichomonas_gallinarum.5730.aa.fasta:67510
#Trichomonas_foetus.PRJNA345179.fasta:25030
#Trichomonas_vaginalis_GenBank.PRJNA16084.fasta:59681
```

```
#Trichomonas_vaginalis_RefSeq.G3.fasta:59679
wc -l *.fasta
#    13116 Dientamoeba_fragilis.43352.aa.fasta
#    13964 Histomonas_meleagridis.135588.aa.fasta
#    89638 Pentatrichomonas_hominis.5728.aa.fasta
#   135020 Tetratrichomonas_gallinarum.5730.aa.fasta
#    50060 Trichomonas_foetus.PRJNA345179.fasta
#   119362 Trichomonas_vaginalis_GenBank.PRJNA16084.fasta
#   119358 Trichomonas_vaginalis_RefSeq.G3.fasta
#   540518 total
#they are now all 1-line FASTAs
###
#now I need to check that the files from Jon are also 1-line
grep -c ">" SC_newprots_may21.fasta
#29853
wc -l SC_newprots_may21.fasta
#352094 SC_newprots_may21.fasta
grep -c ">" BM_newprots_may21.fasta
#14817
wc -l BM_newprots_may21.fasta
#161786 BM_newprots_may21.fasta
grep -c ">" BS_newprot_may21.fasta
#29791
wc -l BS_newprot_may21.fasta
#346782 BS_newprot_may21.fasta
#so no, these are multi-line FASTAs
#time to turn them into 1-line FASTAs
awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}  END {printf("\n");}' < SC_newprots_may21.fasta \
  > SC_newprots_may21.anaeramoeba_start.fasta
awk '{if (NR==1 && NF==0) next};1' SC_newprots_may21.anaeromoeba_start.fasta > SC_newprots_may21.anaeromoeba.fasta
awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}  END {printf("\n");}' < BM_newprots_may21.fasta \
  > BM_newprots_may21.anaeromoeba_start.fasta
awk '{if (NR==1 && NF==0) next};1' BM_newprots_may21.anaeromoeba_start.fasta > BM_newprots_may21.anaeromoeba.fasta
awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}  END {printf("\n");}' < BS_newprot_may21.fasta \
  > BS_newprots_may21.anaeromoeba_start.fasta
awk '{if (NR==1 && NF==0) next};1' BS_newprots_may21.anaeromoeba_start.fasta > BS_newprots_may21.anaeromoeba.fasta
#final checks:
grep -c ">" *.fasta
#BM_newprots_may21.anaeromoeba.fasta:14817
#BS_newprots_may21.anaeromoeba.fasta:29791
#Dientamoeba_fragilis.43352.aa.fasta:6558
#Histomonas_meleagridis.135588.aa.fasta:6982
#Pentatrichomonas_hominis.5728.aa.fasta:44819
#SC_newprots_may21.anaeromoeba.fasta:29853
#Tetratrichomonas_gallinarum.5730.aa.fasta:67510
#Trichomonas_foetus.PRJNA345179.fasta:25030
#Trichomonas_vaginalis_GenBank.PRJNA16084.fasta:59681
#Trichomonas_vaginalis_RefSeq.G3.fasta:59679
wc -l *.fasta
#    29634 BM_newprots_may21.anaeromoeba.fasta
#    59582 BS_newprots_may21.anaeromoeba.fasta
#    13116 Dientamoeba_fragilis.43352.aa.fasta
#    13964 Histomonas_meleagridis.135588.aa.fasta
#    89638 Pentatrichomonas_hominis.5728.aa.fasta
#    59706 SC_newprots_may21.anaeromoeba.fasta
#   135020 Tetratrichomonas_gallinarum.5730.aa.fasta
#    50060 Trichomonas_foetus.PRJNA345179.fasta
#   119362 Trichomonas_vaginalis_GenBank.PRJNA16084.fasta
#   119358 Trichomonas_vaginalis_RefSeq.G3.fasta
#   689440 total
#looks good!
```

Python FASTA editing scripts

This Python script replaces the FASTA headers with random alphanumeric sequences that are 16 characters long, and generates a reference file so the process can be traced back. This code is in the assignFASTAheaders.py file.

```
#!/bin/python
"""
```

```
Title: assignFASTAheaders.py
Date: 28.07.2021
Author: Virág Varga

Description:
    This program replaces FASTA headers in a FASTA file with a 16-character random
        alphanumeric code. A reference file is also printed that links the
        random code to the original FASTA header.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    re
    random
    string

Procedure:
    1. Loading required modules & assigning command line argument.
    2. Parsing the input FASTA file in order to extract headers and generate
        random alphanumeric codes to replace them.
    3. Writing out the new FASTA file with the alphanumeric code headers,
        accompanied by the reference file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.

Usage
    ./assignFASTAheaders.py input_fasta
    OR
    python assignFASTAheaders.py input_fasta

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""

#import necessary modules
import sys #allows execution of script from command line
import re #enables regex pattern matching
import random #enables random number & variable generation
import string #imports a collection of string constants


#load input and output files
input_fasta = sys.argv[1]
#input_fasta = "D_fragilis.20test.fasta"
output_fasta = ".".join(input_fasta.split('.')[:-1]) + '_edit.fasta'
ref_doc = ".".join(input_fasta.split('.')[:-1]) + '_ref.txt'


#write the program
with open(input_fasta, "r") as infile, open(output_fasta, "w") as outfile, open(ref_doc, "w") as outref:
    #open the input and output fasta files
    for line in infile:
        #iterate through the input file line by line
        if line.startswith(">"):
            #identify the header lines
            header = line
            #remove the ">" character at the start of the line
            #this enables easier manipulation of the FASTA header
            header = re.sub(">", "", header)
            #generate a random 16-character alphanumeric string to replace the original header
            assigned_header = ''.join(random.choices(string.ascii_letters + string.digits, k=16))
            #now print the new header to the outfile
            outfile.write(">" + assigned_header + "\n")
            #and print the assigned reference to the outref file
            outref.write(assigned_header + "\t" + header)
        else:
            #sequence lines are copied to the outfile without changes
            outfile.write(line)
```

Now let's test that the program runs from the command line as intended:

```
python assignFASTAheaders.py D_fragilis.20test.fasta
#it worked
```

Now let's write a program to reverse the process. This program is in the file reverseHeaders.py

```
#!/bin/python
"""

Title: reverseHeaders.py
Date: 28.07.2021
Author: Virág Varga

Description:
    This program replaces the random alphanumeric headers assigned by the
        assignFASTAheaders.py program with the original FASTA headers, using the
        reference file created by the other program as a guide.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    re
    pandas

Procedure:
    1. Loading required modules & assigning command line arguments.
    2. Using Pandas to load the contents of the reference file into a dictionary.
    3. Parsing the input FASTA file in order to match the random headers to the
        original headers via the reference file.
    4. Writing out the new FASTA file with the original FASTA headers.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - The user needs to check that the correct reference file is assigned.

Usage
    ./reverseHeaders.py input_fasta ref_doc
    OR
    python reverseHeaders.py input_fasta ref_doc

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""

#import necessary modules
import sys #allows execution of script from command line
import re #enables regex pattern matching
import pandas as pd #allow manipulation of data files


#load input and output files
input_fasta = sys.argv[1]
#input_fasta = "D_fragilis.20test_edit.fasta"
#the user choses the specific file, in case it isn't the original FASTA being used for the reversal
ref_doc = sys.argv[2]
#ref_doc = "D_fragilis.20test_ref.txt"
#output_fasta name is based on the input_fasta name
output_fasta = ".".join(input_fasta.split('.')[:-1]) + '_ogs.fasta'


#create and populate dictionary using the reference file
with open(ref_doc, "r") as inref:
    REF = pd.read_csv(inref, sep="\t", header=None)
    REF = REF.set_index(0)
    ref_dict = REF.T.to_dict('list')
    #note that this manner of conversion makes the dictionary value a list


#write the program
```

```
with open(input_fasta, "r") as infile, open(output_fasta, "w") as outfile:
    #open the input and output fasta files
    for line in infile:
        #iterate through the input file line by line
        if line.startswith(">"):
            #identify the header lines
            header = line.strip()
            #remove the ">" character at the start of the line
            #this enables easier manipulation of the FASTA header
            header = re.sub(">", "", header)
            #iterate through the reference dictionary keys
            for k in ref_dict.keys():
                #print(k)
                if header == k:
                    og_header = ref_dict[k]
                    #the og_header will be in list form, so need to convert to string
                    og_header2 = ' '.join([str(item) for item in og_header])
                    #now print the new header to the outfile
                    outfile.write(">" + og_header2 + "\n")
        else:
            #sequence lines are copied to the outfile without changes
            outfile.write(line)
```

Check that it works from the command line:

```
python reverseHeaders.py D_fragilis.20test_edit.fasta D_fragilis.20test_ref.txt
#it worked
```

So now to encode/convert all data files:

```
#let's try to speed this up by doing it all in a loop
ls MainData/*.fasta | while read file; do
  python assignFASTAheaders.py $file
done
#now let's move everything to the proper folders
mv MainData/*_edit.fasta DataEncoding/
mv MainData/*_ref.txt DataEncoding/
#now make symbolic link in EncodedData/ directory
#working from main Trich_Parab/ directory
ln -s /home/inf-47-2020/Trich_Parab/Data_Files/DataEncoding/*_edit.fasta EncodedData/
```

Orthologous Clustering & Secretome Identification

OrthoFinder

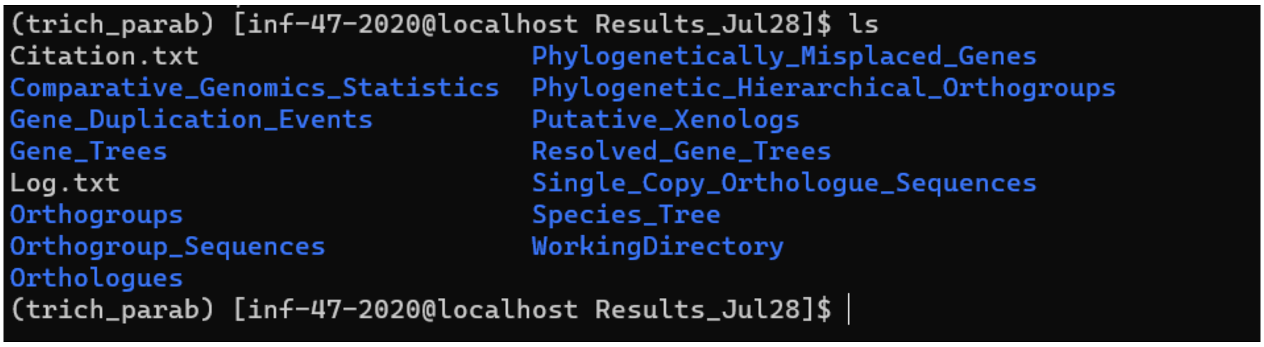
Exploring the program, based on tutorial, here: https://davidemms.github.io/menu/tutorials.html

```
#accessing the program help
orthofinder -h
#other aspects of the tutorial don't all work,
#because installation with conda doesn't download sample files
```

Running the program:

```
#running the program
orthofinder -f EncodedData/ -a 4 -o OrthoFinderResults/
#`-f` gives the directory where the analyzed files are
#`-a` determines number of threads (default 1)
```

```
#`-o` non-default results directory (program makes it, can't be extant)
#results are in OrthoFinderResults/Results_Jul28/
```



```
(trich_parab) [inf-47-2020@localhost Results_Jul28]$ ls
Citation.txt                     Phylogenetically_Misplaced_Genes
Comparative_Genomics_Statistics  Phylogenetic_Hierarchical_Orthogroups
Gene_Duplication_Events          Putative_Xenologs
Gene_Trees                       Resolved_Gene_Trees
Log.txt                          Single_Copy_Orthologue_Sequences
Orthogroups                      Species_Tree
Orthogroup_Sequences             WorkingDirectory
Orthologues
(trich_parab) [inf-47-2020@localhost Results_Jul28]$ 
```

Results of analysis (with tutorial as guide, here: https://davidemms.github.io/orthofinder_tutorials/exploring-orthofinders-results.html):

- Percentage of genes in orthogroups: 86.9
  - This is good; the value is should be >80%
  - When broken down to the species level, most of the species stay good, with only Histomonas meleagridis, Tetratrichomonas_gallinarum and Trichomonas_foetus being just a little below, in the 70-80% range.
- Phylogenetic tree: This looks good! It seemed suspect to me at first, but Courtney says it's correct.
  - **Note**: The cause of the issue was a problem with naming. It's not *Trichomonas foetus*, but *Tritrichomonas foetus*.
  - Files: SpeciesTree_rooted.txt & SeciesTree_rooted_node_labels.txt
- Most important files:
  - Statistics_Overall.tsv
  - Statistics_PerSpecies.tsv
  - Orthogroups.tsv

SignalP

Exploring the program:

```
#bring up the manual on the terminal
signalp -h
```

Running the program:

```
#run SignalP in a loop
ls ../EncodedData/*.fasta | while read file; do
  signalp -format 'long' -org 'euk' -plot 'none' -fasta $file
done
#this program makes a lot of files for each species,
#so these files should be separated out into folders by species
#use a series of `mv` commands
#folder creation
mkdir BM_newprots_anaeromoeba_SP BS_newprots_anaeromoeba_SP D_fragilis_SP H_meleagridis_SP P_hominis_SP \
  SC_newprots_anaeromoeba_SP T_gallinarum_SP T_foetus_SP T_vaginalis_GB_SP T_vaginalis_RS_SP
#moving files
mv BM_newprots_may21.anaeromoeba* BM_newprots_anaeromoeba_SP
mv BS_newprots_may21.anaeromoeba* BS_newprots_anaeromoeba_SP
mv Dientamoeba_fragilis.43352.aa* D_fragilis_SP
mv Histomonas_meleagridis.135588.aa* H_meleagridis_SP
mv Pentatrichomonas_hominis.5728.aa* P_hominis_SP
#-bash: /usr/bin/mv: Argument list too long
mv SC_newprots_may21.anaeromoeba* SC_newprots_anaeromoeba_SP
mv Tetratrichomonas_gallinarum.5730.aa* T_gallinarum_SP
#-bash: /usr/bin/mv: Argument list too long
mv Trichomonas_foetus.PRJNA345179* T_foetus_SP
mv Trichomonas_vaginalis_GenBank* T_vaginalis_GB_SP
#-bash: /usr/bin/mv: Argument list too long
mv Trichomonas_vaginalis_RefSeq.G3* T_vaginalis_RS_SP
#-bash: /usr/bin/mv: Argument list too long
#extract the summary files for each run using
find . -name *summary*
#"." tells `find` which directory to look in
#`-name` tells `find` to search by the content of the file name
#copy the file out to the larger SignalP directory with `cp`
#tar the files since they're so large
tar -zcvf BM_newprots_anaeromoeba_SP.tar.gz BM_newprots_anaeromoeba_SP/
#to untar, use:
tar -zxvf BM_newprots_anaeromoeba_SP.tar.gz
#now to do the rest
tar -zcvf BS_newprots_anaeromoeba_SP.tar.gz BS_newprots_anaeromoeba_SP/
tar -zcvf D_fragilis_SP.tar.gz D_fragilis_SP/
tar -zcvf H_meleagridis_SP.tar.gz H_meleagridis_SP/
tar -zcvf SC_newprots_anaeromoeba_SP.tar.gz SC_newprots_anaeromoeba_SP/
tar -zcvf T_foetus_SP.tar.gz T_foetus_SP/
#now let's deal with the ones that didn't work:
#P. hominis, T. gallinarum, & the 2 T. vaginalis types
#for these, delete file files and run again
#rerunning SignalP for the 4 files, in their final directories
signalp -format 'long' -org 'euk' -plot 'none' -fasta ../../EncodedData/Pentatrichomonas_hominis.5728.aa_edit.fasta
tar -zcvf P_hominis_SP.tar.gz P_hominis_SP/
signalp -format 'long' -org 'euk' -plot 'none' -fasta ../../EncodedData/Tetratrichomonas_gallinarum.5730.aa_edit.fasta
tar -zcvf T_gallinarum_SP.tar.gz T_gallinarum_SP/
signalp -format 'long' -org 'euk' -plot 'none' \
  -fasta ../../EncodedData/Trichomonas_vaginalis_GenBank.PRJNA16084_edit.fasta
tar -zcvf T_vaginalis_GB_SP.tar.gz T_vaginalis_GB_SP/
signalp -format 'long' -org 'euk' -plot 'none' -fasta ../../EncodedData/Trichomonas_vaginalis_RefSeq.G3_edit.fasta
tar -zcvf T_vaginalis_RS_SP.tar.gz T_vaginalis_RS_SP/
#and then delete the original directories
rm -r dir1
#adapted:
rm -r BM_newprots_anaeromoeba_SP
rm -r SC_newprots_anaeromoeba_SP
rm -r BS_newprots_anaeromoeba_SP
rm -r T_foetus_SP
rm -r D_fragilis_SP
rm -r T_gallinarum_SP
rm -r H_meleagridis_SP
rm -r T_vaginalis_GB_SP
rm -r T_vaginalis_RS_SP
rm -r P_hominis_SP
#done!
ls
#BM_newprots_anaeromoeba_SP.tar.gz  SC_newprots_anaeromoeba_SP.tar.gz
#BS_newprots_anaeromoeba_SP.tar.gz  T_foetus_SP.tar.gz
#D_fragilis_SP.tar.gz               T_gallinarum_SP.tar.gz
#H_meleagridis_SP.tar.gz            T_vaginalis_GB_SP.tar.gz
#P_hominis_SP.tar.gz                T_vaginalis_RS_SP.tar.gz
```

eggNOG

Running the program:

```
#based on the exercise file
#working in the Trich_Parab/EggNOGResults/ directory
# Make a temporary folder in case something goes wrong in your run
mkdir OG_comp_tmp
emapper.py -m diamond -i ../EncodedData/BM_newprots_may21.anaeromoeba_edit.fasta --report_orthologs \
  --pfam_realign realign -o BM_anaeromoeba_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/BS_newprots_may21.anaeromoeba_edit.fasta --report_orthologs \
  --pfam_realign realign -o BS_anaeromoeba_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Dientamoeba_fragilis.43352.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o D_fragilis_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Histomonas_meleagridis.135588.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o H_meleagridis_EN.emap --temp_dir OG_comp_tmp --cpu 2
```

```
emapper.py -m diamond -i ../EncodedData/Pentatrichomonas_hominis.5728.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o P_hominis_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/SC_newprots_may21.anaeromoeba_edit.fasta --report_orthologs \
  --pfam_realign realign -o SC_anaeromoeba_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Tetratrichomonas_gallinarum.5730.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_gallinarum_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Trichomonas_foetus.PRJNA345179_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_foetus_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Trichomonas_vaginalis_GenBank.PRJNA16084_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_vaginalis_GB_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Trichomonas_vaginalis_RefSeq.G3_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_vaginalis_RS_EN.emap --temp_dir OG_comp_tmp --cpu 2
#done
#stdout saved to program_printouts2.txt
```

Program print-outs copied to: program_printouts2.txt

DeepLoc

Exploring the program:

```
#activate the necessary conda environment
conda activate DeepLoc_usage
#there doesn't seem to be a manual
#only the README
#I'm struggling to find example code, too
#so we'll see how this goes
```

Running the program:

```
# running the code the way it should work:
#looping would be nicer, but then I can't rename the files then
deeploc --fasta ../../EncodedData/BM_newprots_may21.anaeromoeba_edit.fasta --output BM_anaeromoeba_DL --attention
#when it finishes, get this printout to stdout:
#WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
#Input file processed. Starting prediction...
#Prediction finished. Generating output...
#Done!
tar -zcvf BM_anaeromoeba_DL.tar.gz BM_anaeromoeba_DL
rm -r BM_anaeromoeba_DL/
deeploc --fasta ../../EncodedData/BS_newprots_may21.anaeromoeba_edit.fasta --output BS_anaeromoeba_DL --attention
#copy out BS_anaeromoeba_DL.txt to the DeepLocResults/ directory
tar -zcvf BS_anaeromoeba_DL.tar.gz BS_anaeromoeba_DL
rm -r BS_anaeromoeba_DL/
deeploc --fasta ../../EncodedData/Dientamoeba_fragilis.43352.aa_edit.fasta --output D_fragilis_DL --attention
#copy out BS_anaeromoeba_DL.txt to the DeepLocResults/ directory
tar -zcvf D_fragilis_DL.tar.gz D_fragilis_DL
rm -r D_fragilis_DL/
deeploc --fasta ../../EncodedData/Histomonas_meleagridis.135588.aa_edit.fasta --output H_meleagridis_DL --attention
#copy out H_meleagridis_DL.txt to the DeepLocResults/ directory
tar -zcvf H_meleagridis_DL.tar.gz H_meleagridis_DL
rm -r H_meleagridis_DL/
deeploc --fasta ../../EncodedData/Pentatrichomonas_hominis.5728.aa_edit.fasta --output P_hominis_DL --attention
#copy out P_hominis_DL.txt to the DeepLocResults/ directory
tar -zcvf P_hominis_DL.tar.gz P_hominis_DL
rm -r P_hominis_DL/
deeploc --fasta ../../EncodedData/SC_newprots_may21.anaeromoeba_edit.fasta --output SC_anaeromoeba_DL --attention
#copy out SC_anaeromoeba_DL.txt to the DeepLocResults/ directory
tar -zcvf SC_anaeromoeba_DL.tar.gz SC_anaeromoeba_DL
rm -r SC_anaeromoeba_DL/
deeploc --fasta ../../EncodedData/Tetratrichomonas_gallinarum.5730.aa_edit.fasta --output T_gallinarum_DL --attention
#copy out T_gallinarum_DL.txt to the DeepLocResults/ directory
tar -zcvf T_gallinarum_DL.tar.gz T_gallinarum_DL
rm -r T_gallinarum_DL/
deeploc --fasta ../../EncodedData/Trichomonas_foetus.PRJNA345179_edit.fasta --output T_foetus_DL --attention
#copy out T_foetus_DL.txt to the DeepLocResults/ directory
```

```
tar -zcvf T_foetus_DL.tar.gz T_foetus_DL
rm -r T_foetus_DL/
deeploc --fasta ../../EncodedData/Trichomonas_vaginalis_GenBank.PRJNA16084_edit.fasta --output T_vaginalis_GB_DL \
  --attention
#copy out T_vaginalis_GB_DL.txt to the DeepLocResults/ directory
tar -zcvf T_vaginalis_GB_DL.tar.gz T_vaginalis_GB_DL
rm -r T_vaginalis_GB_DL/
deeploc --fasta ../../EncodedData/Trichomonas_vaginalis_RefSeq.G3_edit.fasta --output T_vaginalis_RS_DL \
  --attention
#copy out T_vaginalis_RS_DL.txt to the DeepLocResults/ directory
tar -zcvf T_vaginalis_RS_DL.tar.gz T_vaginalis_RS_DL
rm -r T_vaginalis_RS_DL/
```

PFam via eggNOG

EggNOG can generate a PFam association for analyzed proteins.

```
#running eggNOG to get teh PFam data
emapper.py -m diamond -i  --report_orthologs --pfam_realign realign -o _EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/BS_newprots_may21.anaeromoeba_edit.fasta --report_orthologs \
  --pfam_realign realign -o BS_anaeromoeba_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Dientamoeba_fragilis.43352.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o D_fragilis_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Histomonas_meleagridis.135588.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o H_meleagridis_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Pentatrichomonas_hominis.5728.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o P_hominis_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/SC_newprots_may21.anaeromoeba_edit.fasta --report_orthologs \
  --pfam_realign realign -o SC_anaeromoeba_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Tetratrichomonas_gallinarum.5730.aa_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_gallinarum_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Trichomonas_foetus.PRJNA345179_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_foetus_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Trichomonas_vaginalis_GenBank.PRJNA16084_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_vaginalis_GB_EN.emap --temp_dir OG_comp_tmp --cpu 2
emapper.py -m diamond -i ../EncodedData/Trichomonas_vaginalis_RefSeq.G3_edit.fasta --report_orthologs \
  --pfam_realign realign -o T_vaginalis_RS_EN.emap --temp_dir OG_comp_tmp --cpu 2
```

Results Filtration & Database Creation

A quick sanity check:

```
#check that none of the proteins actually have the same code
#they really shouldn't, but just in case:
#psuedocode to adapt:
cat index files | cut -f 1 | sort | uniq | wc -l
#If line count == wc -l cat index files, then you have no repeats
#adapting it:
#working in Trich_Parab/Data_Files/DataEncoding directory
cat *_ref.txt > encoding_summary_ref.txt
#basic checks
wc -l *_ref.txt
#   14817 BM_newprots_may21.anaeromoeba_ref.txt
#   29791 BS_newprots_may21.anaeromoeba_ref.txt
#    6558 Dientamoeba_fragilis.43352.aa_ref.txt
#  344720 encoding_summary_ref.txt
#    6982 Histomonas_meleagridis.135588.aa_ref.txt
#   44819 Pentatrichomonas_hominis.5728.aa_ref.txt
#   29853 SC_newprots_may21.anaeromoeba_ref.txt
#   67510 Tetratrichomonas_gallinarum.5730.aa_ref.txt
#   25030 Trichomonas_foetus.PRJNA345179_ref.txt
#   59681 Trichomonas_vaginalis_GenBank.PRJNA16084_ref.txt
#   59679 Trichomonas_vaginalis_RefSeq.G3_ref.txt
#  689440 total
#I did the math, and those line numbers work out
```

```
cat encoding_summary_ref.txt | cut -f 1 | sort | uniq | wc -l
#344720
#which is exactly what it should be, equal to the file line number
#the chances that there would be a repeat of a code would be
#1/108160=9.246e-6
#literally astronomical odds
```

Selecting columns

Working with Pandas Array (this script saved as extractOGs.py):

```
#!/bin/python
"""

Title: extractOGs.py
Date: 09.08.2021
Author: Virág Varga

Description:
    This program parses the Orthogroups.tsv file (an output of OrthoFinder), and
        creates 2 output files: a file containing all rows where all columns
        contain data, and a file containing all rows containing only data unique
        to the Trichomonas vaginalis proteome.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    pandas

Procedure:
    1. Loading required module; defining inputs and outputs.
    2. Using Pandas import the contents of the Orthogroups.tsv file into a
        Pandas dataframe.
    3. Subsetting a dataframe that contains only the rows where all species
        have proteins pretein in the OG; this dataframe is written out to the
        All_species_OGs.csv file.
    4. Subsetting a dataframe that contains only the rows where both versions of
        T. vaginalis (and no other species) have proteins present in the OG;
        this dataframe is written out to the Tvag_unique_OGs.csv file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program can only be run from a directory containing 1 copy of the
        Orthogroups.tsv output file of the OrthoFinder program.
    - Input and output files are not user-defined.

Usage
    ./extractOGs.py
    OR
    python extractOGs.py

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import pandas as pd


#define inputs and outputs
ortho_input = "Orthogroups.tsv"
#ortho_input = "Orthogroups_extract.tsv"
full_output = "All_species_OGs.csv"
#full_output = "All_species_OGs_extract.csv"
Tvag_output = "Tvag_unique_OGs.csv"
#Tvag_output = "Tvag_unique_OGs_extract.csv"
#.csv files are used as outfiles for ease of import into R during the final data analysis stage
```

```
#read in the input tsv file, assigning the first row as a header row
ortho_df = pd.read_csv(ortho_input, sep='\t', header=0)
#set the first column (containing the OGs) as an index
ortho_df.set_index('Orthogroup')


#keep only rows that contain no empty columns:
with open(full_output, "w") as outfile1:
    #open the appropriate outfile for writing
    #subset a dataframe containing only rows where each species has an entry for the OG
    full_ortho_df = ortho_df.dropna()
    #write out the full_ortho_df dataframe to the outfile
    full_ortho_df.to_csv(outfile1, index = False, line_terminator='\n')


#keep only rows that have data in both T. vaginalis columns:
with open(Tvag_output, "w") as outfile2:
    #open the appropriate outfile for writing
    #identify rows where both versions of T. vaginalis have entries for the OG
    Tvag_df1 = ortho_df[ortho_df['Trichomonas_vaginalis_GenBank.PRJNA16084_edit'].notnull() &
    ortho_df['Trichomonas_vaginalis_RefSeq.G3_edit'].notnull()]
    #from the subset above, identify rows where ONLY T. vaginalis has entries for the OG
    Tvag_df2 = Tvag_df1[Tvag_df1['BM_newprots_may21.anaeromoeba_edit'].isnull() &
    Tvag_df1['BS_newprots_may21.anaeromoeba_edit'].isnull() &
    Tvag_df1['Dientamoeba_fragilis.43352.aa_edit'].isnull() &
    Tvag_df1['Histomonas_meleagridis.135588.aa_edit'].isnull() &
    Tvag_df1['Pentatrichomonas_hominis.5728.aa_edit'].isnull() &
    Tvag_df1['SC_newprots_may21.anaeromoeba_edit'].isnull() &
    Tvag_df1['Tetratrichomonas_gallinarum.5730.aa_edit'].isnull() &
    Tvag_df1['Trichomonas_foetus.PRJNA345179_edit'].isnull()]
    #write out the Tvag_df2 dataframe to the outfile
    Tvag_df2.to_csv(outfile2, index = False, line_terminator='\n')
```

Compiling a database

Python code to extract PFAM data from eggNOG results (saved in file eggNOG_PFam_Parser.py):

```
#!/bin/python
"""
Title: eggNOG_PFam_Parser.py
Date: 2021-08-09
Authors: Virág Varga

Description:
    This program parses the PFam search results froduced by the eggNOG program
        and creates an output text file containing selected categories of information
        for each query sequence.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys

Procedure:
    1. Assigning command line arguments and output file name; loading module.
    2. Creating a dictionary containing the pertinent results of the eggNOG PFam
        search file.
    3. Writing out the results to a tab-separated text file.

Known bugs and limitations:
    - This eggNOG PFam results parser is made specifically to suit the formatting
        of the eggNOG PFam search output files.
    - There is no quality-checking integrated into the code.
    - The name of the output file is not user-defined.

Usage
    ./eggNOG_PFam_Parser.py input_file
    OR
    python eggNOG_PFam_Parser.py input_file

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""

#Part 1: Setup

#import necessary modules
import sys #allow assignment of files from the command line
import os #allow access to computer files


#assign command line argument
input_file = sys.argv[1]
#input_file = "BM_anaeromoeba_EN.emap.emapper.pfam"
base = os.path.basename(input_file)
out_full = os.path.splitext(base)[0]
outname = out_full.split(".")[0]
output_file = outname + "_PFam.txt"
#output_file = "BM_anaeromoeba_PFam.txt"


#Part 2: Parse through the file to create dictionary with desired information
#Part 3: Write out the results to the output file

#create empty dictionary to contain the data that will be written out
target_dict = {}

with open(input_file, "r") as infile, open(output_file, "w") as outfile:
    #open the input BLASTp result file for reading
    #open the output file for writing
    for line in infile:
        #read through the file line by line
        seq_list = [None] * 3
        #empty list seq_list will hold the alignment data associated with each query-target match
        #with each iteration of the loop, this list is overwritten
        if line.startswith('# #query_name'):
            #identify lines that start with "# #query_name" - these lines contain the query sequences
            pass
            #pass to the next line, where the query proteins will start
        elif not line.startswith("#"):
            #identify lines that start with ">" character - these contain target sequence names
            query_results = line.split()
            #the line with the query id is 'split' - separated into a list based on the locations of spaces
            query_id = query_results[0]
            #the item with index 0 in the query_results list is the query sequence name
            #the query sequence name is placed into the variable query_id
            pfam_id = query_results[1]
            #variable pfam_id contains the pfam hit for the query sequence
            query2hit = query_id + "\t" + pfam_id
            #variable query2hit contains a string with the query_id and pfam_id separated by a tab ("\t")
            seq_list[0] = query_results[2]
            #index 0 of seq_list is the e-value
            seq_list[1] = query_results[3]
            #index 1 of seq_list is the sum score
            seq_list[2] = query_results[9]
            #index 2 of seq_list is the query coverage
            target_dict[query2hit] = seq_list
            #the query2hit variable is used as the key to the target_dict dictionary
            #the associated value is the contents of list seq_list
    outfile.write("Query" + "\t" + "Hit/PFam" + "\t" + "E-Value" + "\t" + "Sum Score" + "\t" + \
    "Query Coverage" + "\n")
    #the header line is prepared and written out to the output file
    for targetKey in target_dict:
        #iterate through the target_dict dictionary using its keys
        outfile.write('{}\t{}\t{}\t{}\n'.format(targetKey, target_dict[targetKey][0], target_dict[targetKey][1], \
    target_dict[targetKey][2]))
        #results are written out to the output file with appropriate formatting
```

Running the above in a loop on the server:

```
#working in the Trich_Parab/EggNOGResults/ directory
ls *.pfam | while read file; do
  python eggNOG_PFam_Parser.py $file;
done
```

Parser for the eggNOG proper results files (saved in eggNOG_Parser.py):

- Should use *.annotations file, and keep pythonic indexes: 0-3, 16, 18, 24
- header/column names: query_name[0] seed_eggNOG_ortholog[1] seed_ortholog_evalue[2] seed_ortholog_score[3] eggNOG[4] OGs[5] narr_og_name[6] narr_og_cat[7] narr_og_desc[8] best_og_name[9] best_og_cat[10] best_og_desc[11] Preferred_name[12] GOs[13] EC[14] KEGG_ko[15] KEGG_Pathway[16] KEGG_Module[17] KEGG_Reaction[18] KEGG_rclass[19] BRITE[20] KEGG_TC[21] CAZy[22] BiGG_Reaction[23] PFAMs[24]
- Keep in mind the other options to save here, though:
  - KEGG: keep pathway & reaction, maybe come back for others if Courtney says
- Don't need to accommodate for null values, because program uses "-"
  - But *will* have to accommodate for this when porting into the large table - will need to convert to "null"

```
#!/bin/python
"""
Title: eggNOG_Parser.py
Date: 2021-08-10
Authors: Virág Varga

Description:
    This program parses the .annotations results file produced by the eggNOG program
        and creates an output text file containing selected categories of information
        for each query sequence.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    os
    csv

Procedure:
    1. Assigning command line arguments and output file name; loading modules.
    2. Creating a dictionary containing the pertinent results of the eggNOG
        annotations file.
    3. Writing out the results to a tab-separated text file.

Known bugs and limitations:
    - This eggNOG results parser is made specifically to suit the formatting
        of the eggNOG annotations output files.
    - There is no quality-checking integrated into the code.
    - The name of the output file is not user-defined.

Usage
    ./eggNOG_Parser.py input_file
    OR
    python eggNOG_Parser.py input_file

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""

#Part 1: Setup

#import necessary modules
import sys #allow assignment of files from the command line
import os #allow access to computer files
import csv #manage the fact that some columns have spaces


#assign command line argument
input_file = sys.argv[1]
```

```
#input_file = "BM_anaeromoeba_EN.emap.emapper.annotations"
base = os.path.basename(input_file)
out_full = os.path.splitext(base)[0]
outname = out_full.split(".")[0]
output_file = outname + "_eggNOG.txt"
#output_file = "BM_anaeromoeba_eggNOG.txt"


#Part 2: Parse through the file to create dictionary with desired information
#Part 3: Write out the results to the output file

#create empty dictionary to contain the data that will be written out
target_dict = {}

with open(input_file, "r") as infile, open(output_file, "w") as outfile:
    #open the input eggNOG result file for reading
    infile_reader = csv.reader(infile, delimiter='\t')
    #opening as csv to allow tab-delimited reading
    #open the output file for writing
    for line in infile_reader:
        #read through the file line by line
        seq_list = [None] * 5
        #empty list seq_list will hold the alignment data associated with each query-target match
        #with each iteration of the loop, this list is overwritten
        if line[0].startswith('#query_name'):
            #identify lines that start with "#query_name" - these lines contain the query sequences
            pass
            #pass to the next line, where the query proteins will start
        elif not line[0].startswith("#"):
            #identify lines that start with ">" character - these contain target sequence names
            query_results = line
            #the list line is saved into variable query_results
            query_id = query_results[0]
            #the item with index 0 in the query_results list is the query sequence name
            #the query sequence name is placed into the variable query_id
            seed_id = query_results[1]
            #variable seed_id contains the eggNOG seed ortholog for the query sequence
            query2hit = query_id + "\t" + seed_id
            #variable query2hit contains a string with the query_id and seed_id separated by a tab ("\t")
            seq_list[0] = query_results[2]
            #index 0 of seq_list is the seed ortholog e-value
            seq_list[1] = query_results[3]
            #index 1 of seq_list is the seed ortholog score
            seq_list[2] = query_results[16]
            #index 2 of seq_list is the KEGG Pathway
            seq_list[3] = query_results[18]
            #index 2 of seq_list is the KEGG Reaction
            seq_list[4] = query_results[23]
            #index 2 of seq_list is the PFams
            target_dict[query2hit] = seq_list
            #the query2hit variable is used as the key to the target_dict dictionary
            #the associated value is the contents of list seq_list
    outfile.write("Query" + "\t" + "Seed Ortholog" + "\t" + "Seed E-Value" + "\t" + "Seed Score" + "\t" + \
    "KEGG Pathway" + "\t" + "KEGG Reaction" + "\t" + "PFams" + "\n")
    #the header line is prepared and written out to the output file
    for targetKey in target_dict:
        #iterate through the target_dict dictionary using its keys
        outfile.write('{}\t{}\t{}\t{}\t{}\t{}\n'.format(targetKey, target_dict[targetKey][0], \
      target_dict[targetKey][1], target_dict[targetKey][2], target_dict[targetKey][3], \
      target_dict[targetKey][4]))
        #results are written out to the output file with appropriate formatting
```

```
#running the program
ls *.annotations | while read file; do
  python eggNOG_Parser.py $file;
done
#then transfer back to the home computer
```

Parser for the SignalP results files (saved in signalP_Parser.py):

- Categories (listed in column 2 (pythonic index [1])) (as per website, here: https://services.healthtech.dtu.dk/service.php?SignalP-5.0) are:
  - Sec/SPI: "standard" secretory signal peptides transported by the Sec translocon and cleaved by Signal Peptidase I (Lep)
  - Sec/SPII: lipoprotein signal peptides transported by the Sec translocon and cleaved by Signal Peptidase II (Lsp)
  - Tat/SPI: Tat signal peptides transported by the Tat translocon and cleaved by Signal Peptidase I (Lep)
  - OTHER
- Want to keep everything except the "other" category (since "other" is non-excreted proteins)
- Want to keep pythonic indices: 0-2
  - where [0] is the ID
  - where [1] is the PREDICTION
  - where [2] is the likelihood that the prediction is correct

```
#!/bin/python
"""
Title: signalP_Parser.py
Date: 2021-08-10
Authors: Virág Varga

Description:
    This program parses the SignalP summary search results and creates an output
        text file containing selected categories of information for each query sequence.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    os

Procedure:
    1. Assigning command line arguments and output file name; loading modules.
    2. Creating a dictionary containing the pertinent results of the SignalP
        search file.
    3. Writing out the results to a tab-separated text file.

Known bugs and limitations:
    - This SignalP results parser is made specifically to suit the formatting
        of the SignalP search output files.
    - There is no quality-checking integrated into the code.
    - The name of the output file is not user-defined.

Usage
    ./signalP_Parser.py input_file
    OR
    python signalP_Parser.py input_file

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""

#Part 1: Setup

#import necessary modules
import sys #allow assignment of files from the command line
import os #allow access to computer files


#assign command line argument
input_file = sys.argv[1]
#input_file = "BM_newprots_may21.anaeromoeba_edit_summary.signalp5"
base = os.path.basename(input_file)
out_full = os.path.splitext(base)[0]
outname = out_full.split(".")[0]
output_file = outname + "_SignalP.txt"
#output_file = "BM_anaeromoeba_SignalP.txt"


#Part 2: Parse through the file to create dictionary with desired information
```

```
#Part 3: Write out the results to the output file

#create empty dictionary to contain the data that will be written out
target_dict = {}

with open(input_file, "r") as infile, open(output_file, "w") as outfile:
    #open the input SignalP result file for reading
    #open the output file for writing
    for line in infile:
        #read through the file line by line
        seq_list = [None] * 2
        #empty list seq_list will hold the alignment data associated with each query-target match
        #with each iteration of the loop, this list is overwritten
        if line.startswith('# ID'):
            #identify lines that start with "# ID" - this is the header line preceding information lines
            pass
            #pass to the next line, where the query proteins will start
        elif not line.startswith("#"):
            #identify lines that don't start with "#" character - these contain target sequence names
            query_results = line.split()
            #the line with the query id is 'split' - separated into a list based on the locations of spaces
            query_id = query_results[0]
            #the item with index 0 in the query_results list is the query sequence name
            #the query sequence name is placed into the variable query_id
            seq_list[0] = query_results[1]
            #index 0 of seq_list is the prediction
            seq_list[1] = query_results[2]
            #index 1 of seq_list is the likelihood that the prediction is correct
            target_dict[query_id] = seq_list
            #the query_id variable is used as the key to the target_dict dictionary
            #the associated value is the contents of list seq_list
    outfile.write("Query" + "\t" + "Prediction" + "\t" + "Probability" + "\n")
    #the header line is prepared and written out to the output file
    for targetKey in target_dict:
        #iterate through the target_dict dictionary using its keys
        outfile.write('{}\t{}\t{}\n'.format(targetKey, target_dict[targetKey][0], target_dict[targetKey][1]))
        #results are written out to the output file with appropriate formatting
```

```
#running the program
ls *.signalp5 | while read file; do
  python signalP_Parser.py $file;
done
#then transfer back to the home computer
```

Parser for the DeepLoc results files (saved in deepLoc_Parser.py):

- Want "Extracellular" in column 2 (pythonic index [1])
    - "Cell_membrane" might also be interesting (% possibility in [7])
- Also want contents of column 6 (pythonic index [5]) which contains the probability of the target being extracellular

```
#!/bin/python
"""
Title: deepLoc_Parser.py
Date: 2021-08-10
Authors: Virág Varga

Description:
    This program parses the DeepLoc search results and creates an output
        text file containing selected categories of information for each query sequence.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    os
```

```
Procedure:
    1. Assigning command line arguments and output file name; loading modules.
    2. Creating a dictionary containing the pertinent results of the DeepLoc
        search file.
    3. Writing out the results to a tab-separated text file.

Known bugs and limitations:
    - This DeepLoc results parser is made specifically to suit the formatting
        of the DeepLoc search output files.
    - There is no quality-checking integrated into the code.
    - The name of the output file is not user-defined.

Usage
    ./deepLoc_Parser.py input_file
    OR
    python deepLoc_Parser.py input_file

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""

#Part 1: Setup

#import necessary modules
import sys #allow assignment of files from the command line
import os #allow access to computer files


#assign command line argument
input_file = sys.argv[1]
#input_file = "BM_anaeromoeba_DL.txt"
base = os.path.basename(input_file)
out_full = os.path.splitext(base)[0]
outname = out_full.split(".")[0]
output_file = outname + "_DeepLoc.txt"
#output_file = "BM_anaeromoeba_DeepLoc.txt"


#Part 2: Parse through the file to create dictionary with desired information
#Part 3: Write out the results to the output file

#create empty dictionary to contain the data that will be written out
target_dict = {}

with open(input_file, "r") as infile, open(output_file, "w") as outfile:
    #open the input DeepLoc result file for reading
    #open the output file for writing
    next(infile)
    #skip first (header) line
    for line in infile:
        #read through the file line by line
        seq_list = [None] * 3
        #empty list seq_list will hold the alignment data associated with each query-target match
        #with each iteration of the loop, this list is overwritten
        query_results = line.split()
        #the line with the query id is 'split' - separated into a list based on the locations of spaces
        query_id = query_results[0]
        #the item with index 0 in the query_results list is the query sequence name
        #the query sequence name is placed into the variable query_id
        seq_list[0] = query_results[1]
        #index 0 of seq_list is the location prediction
        seq_list[1] = query_results[5]
        #index 1 of seq_list is the likelihood that the protein target is extracellular
        seq_list[2] = query_results[7]
        #index 2 of seq_list is the likelihood that the protein target is the cell membrane
        target_dict[query_id] = seq_list
        #the query_id variable is used as the key to the target_dict dictionary
        #the associated value is the contents of list seq_list
    outfile.write("Query" + "\t" + "Location Prediction" + "\t" + "Extracellular Probability" + "\t" + \
    "Cell Membrane Probability" + "\n")
    #the header line is prepared and written out to the output file
    for targetKey in target_dict:
        #iterate through the target_dict dictionary using its keys
        outfile.write('{}\t{}\t{}\t{}\n'.format(targetKey, target_dict[targetKey][0], target_dict[targetKey][1], \
    target_dict[targetKey][2]))
        #results are written out to the output file with appropriate formatting
```

```
#running the program
ls *_DL.txt | while read file; do
  python deepLoc_Parser.py $file;
done
#then transfer back to the home computer
```

Script combing the parsed data files into 1 large "database" per species (saved in combo_OG_results.py):

- Import parsed data files as Pandas dataframes, then directly into dictionaries
  - The query ids should be the keys in all dictionaries
  - Need to condense results from ex. the eggNOG PFam results, so don't have multiple rows for the same protein (because that would mess with the other parsed data files)
- Use `sys.argv` to manually select from the command line the parsed files to be used
- Results file should have query id in first column (pythonic index [0]), and all other data in successive columns
  - need to include an option for a null value in cases without a match (ex. "other" target in SignalP, no PFam hits, etc.)
- Column headers: "Query"[0] + "\t" + "PFam Hit"[1] + "\t" + "E-Value"[2] + "\t" + "Sum Score"[3] + "\t" + "Query Coverage"[4] + "\t" + "eggNOG Seed Ortholog"[5] + "\t" + "Seed E-Value"[6] + "\t" + "Seed Score"[7] + "\t" + "KEGG Pathway"[8] + "\t" + "KEGG Reaction"[9] + "\t" + "PFams"[10] + "\t" + "DeepLoc Location Prediction"[11] + "\t" + "Extracellular Probability"[12] + "\t" + "SignalP Prediction"[13] + "\t" + "Probability"[14] + "\n"

```
#!/bin/python
"""

Title: combo_OG_results.py
Date: 10.08.2021
Author: Virág Varga

Description:
    This program parses the results files for 1 species from the eggNOG_PFam_Parser.py,
        eggNOG_Parser.py, signalP_Parser.py and deepLoc_Parser.py scripts. The data in
        these parsed files is concatenated into one large, flat database.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas
    numpy

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the eggNOG_PFam_Parser.py, eggNOG_Parser.py,
        signalP_Parser.py and deepLoc_Parser.py results files.
    3. Creating a list containing all of the protein query ids present in the results
        files for the species, without replicates. Creating an empty dictionary that
        will be populated by the data from the results files.
    4. Iterating through the query protein ids and populating the dictionary with the
        query ids as the keys and the rest of the data as the values in a list.
    5. Writing out the results to a tab-delimited text file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of the parsed files created thorugh the use of
        eggNOG_PFam_Parser.py, eggNOG_Parser.py, signalP_Parser.py and deepLoc_Parser.py.
    - All input and output files are user-defined: This means the user must ensure that
        the correct file names have been assigned to the program.

Usage
    ./combo_OG_results.py PFam_EN_Parsed EggNOG_Parsed DeepLoc_Parsed SignalP_Parsed Output_DB
    OR
    python combo_OG_results.py PFam_EN_Parsed EggNOG_Parsed DeepLoc_Parsed SignalP_Parsed Output_DB

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python
import numpy as np #facilitates manipulation of arrays in Python


#assign command line arguments: input & output files
PFam_EN_Parsed = sys.argv[1]
#PFam_EN_Parsed = "BM_anaeromoeba_EN_PFam.txt"
EggNOG_Parsed = sys.argv[2]
#EggNOG_Parsed = "BM_anaeromoeba_EN_eggNOG.txt"
DeepLoc_Parsed = sys.argv[3]
#DeepLoc_Parsed = "BM_anaeromoeba_DL_DeepLoc.txt"
SignalP_Parsed = sys.argv[4]
#SignalP_Parsed = "BM_newprots_may21_SignalP.txt"
Output_DB = sys.argv[5]
#Output_DB = "BM_anaeromoeba_DB_TEST.txt"


#creating a set of functions to extract the data from the various input files
#PFam results
with open(PFam_EN_Parsed, "r") as pfam_infile:
    #open the parsed PFam via eggNOG data file
    pfam_df = pd.read_csv(pfam_infile, sep='\t', header = 0)
    #read the file into a pandas dataframe
    #specifying that the file is tab-separated with a header line
    pfam_df.set_index('Query')
    #set the first column (containing query sequence names) as an index
    pfam_df = pfam_df.replace(r'^\s*$', np.NaN, regex=True)
    #replace empty cells with 'None'
    pfam_query_ids = pfam_df['Query'].tolist()
    #create a list of the query sequence ids present in the file

#eggNOG results
with open(EggNOG_Parsed, "r") as eggnog_infile:
    #open the parsed eggNOG data file
    eggnog_df = pd.read_csv(eggnog_infile, sep='\t', header = 0)
    #read the file into a pandas dataframe
    #specifying that the file is tab-separated with a header line
    eggnog_df.set_index('Query')
    #set the first column (containing query sequence names) as an index
    eggnog_df = eggnog_df.replace(r'^\s*$', np.NaN, regex=True)
    #replace empty cells with 'None'
    eggnog_query_ids = eggnog_df['Query'].tolist()
    #create a list of the query sequence ids present in the file

#DeepLoc results
with open(DeepLoc_Parsed, "r") as deeploc_infile:
    #open the parsed DeepLoc data file
    deeploc_df = pd.read_csv(deeploc_infile, sep='\t', header = 0)
    #read the file into a pandas dataframe
    #specifying that the file is tab-separated with a header line
    deeploc_df.set_index('Query')
    #set the first column (containing query sequence names) as an index
    deeploc_df = deeploc_df.replace(r'^\s*$', np.NaN, regex=True)
    #replace empty cells with 'None'
    deeploc_query_ids = deeploc_df['Query'].tolist()
    #create a list of the query sequence ids present in the file

#SignalP results
with open(SignalP_Parsed, "r") as signalp_infile:
    #open the parsed SignalP data file
    signalp_df = pd.read_csv(signalp_infile, sep='\t', header = 0)
    #read the file into a pandas dataframe
    #specifying that the file is tab-separated with a header line
    signalp_df.set_index('Query')
    #set the first column (containing query sequence names) as an index
    signalp_df = signalp_df.replace(r'^\s*$', np.NaN, regex=True)
    #replace empty cells with 'None'
    signalp_query_ids = signalp_df['Query'].tolist()
    #create a list of the query sequence ids present in the file


#populating a new list with the data from above
query_ids_list = pfam_query_ids + eggnog_query_ids + deeploc_query_ids + signalp_query_ids
#concatenate the query id lists into 1 large list
#now make a version with no duplicates
query_ids_no2 = []
#empty list query_ids_no2 will contain the non-duplicate list of of query ids
for i in query_ids_list:
    #iterate through the list of query ids
    if i not in query_ids_no2:
        #any query ids that are not yet in list query_ids_no2 are appended to the list
        query_ids_no2.append(i)

#creating an empty dictionary to populate with the parsed data
query_dict = {}


#the main body of the program: creating the species database
with open(Output_DB, "w") as outfile_db:
    #open the outfile for writing
    #populate the query_dict dictionary with the contents of the various dataframes
    for prot_q in query_ids_no2:
        #iterate through the query ids present in list query_ids_no2
        seq_list = [None] * 15
        #empty list seq_list will hold the data associated with each query-data match
        #with each iteration of the loop, this list is overwritten
        if prot_q in pfam_df.values:
            #check if the prot_q protein id is in the pfam_df dataframe
            seq_list[0] = pfam_df.loc[pfam_df['Query'] == prot_q]['Hit/PFam'].values[0]
            #in the row where the query id (column with the header 'Query')
            #where the query matches prot_q
            #the data in the cell of the 'Hit/PFam' column is extracted
            #`.values[0]` indexing is used to prevent the result from printing as an array
            seq_list[1] = pfam_df.loc[pfam_df['Query'] == prot_q]['E-Value'].values[0]
            #the data in the cell of the 'E-Value' column is extracted
            seq_list[2] = pfam_df.loc[pfam_df['Query'] == prot_q]['Sum Score'].values[0]
            #the data in the cell of the 'Sum Score' column is extracted
            seq_list[3] = pfam_df.loc[pfam_df['Query'] == prot_q]['Query Coverage'].values[0]
            #the data in the cell of the 'Query Coverage' column is extracted
        if prot_q in eggnog_df.values:
            #check if the prot_q protein id is in the eggnog_df dataframe
            seq_list[4] = eggnog_df.loc[eggnog_df['Query'] == prot_q]['Seed Ortholog'].values[0]
            #where the query id in prot_q finds a match in the 'Query' column of eggnog_df
            #the data in the cell of the 'Seed Ortholog' column is extracted
            seq_list[5] = eggnog_df.loc[eggnog_df['Query'] == prot_q]['Seed E-Value'].values[0]
            #the data in the cell of the 'Seed E-Value' column is extracted
            seq_list[6] = eggnog_df.loc[eggnog_df['Query'] == prot_q]['Seed Score'].values[0]
            #the data in the cell of the 'Seed Score' column is extracted
            seq_list[7] = eggnog_df.loc[eggnog_df['Query'] == prot_q]['KEGG Pathway'].values[0]
            #the data in the cell of the 'KEGG Pathway' column is extracted
            seq_list[8] = eggnog_df.loc[eggnog_df['Query'] == prot_q]['KEGG Reaction'].values[0]
            #the data in the cell of the 'KEGG Reaction' column is extracted
            seq_list[9] = eggnog_df.loc[eggnog_df['Query'] == prot_q]['PFams'].values[0]
            #the data in the cell of the 'PFams' column is extracted
        if prot_q in deeploc_df.values:
            #check if the prot_q protein id is in the deeploc_df dataframe
            seq_list[10] = deeploc_df.loc[deeploc_df['Query'] == prot_q]['Location Prediction'].values[0]
            #where the query id in prot_q finds a match in the 'Query' column of deeploc_df
            #the data in the cell of the 'Location Prediction' column is extracted
            seq_list[11] = deeploc_df.loc[deeploc_df['Query'] == prot_q]['Extracellular Probability'].values[0]
            #the data in the cell of the 'Extracellular Probability' column is extracted
            seq_list[12] = deeploc_df.loc[deeploc_df['Query'] == prot_q]['Cell Membrane Probability'].values[0]
            #the data in the cell of the 'Extracellular Probability' column is extracted
        if prot_q in signalp_df.values:
            #check if the prot_q protein id is in the signalp_df dataframe
            seq_list[13] = signalp_df.loc[signalp_df['Query'] == prot_q]['Prediction'].values[0]
            #where the query id in prot_q finds a match in the 'Query' column of signalp_df
            #the data in the cell of the 'Prediction' column is extracted
            seq_list[14] = signalp_df.loc[signalp_df['Query'] == prot_q]['Probability'].values[0]
            #the data in the cell of the 'Probability' column is extracted
        query_dict[prot_q] = seq_list
        #populate query_dict dictionary using the protein id in prot_q as the key and the list seq_list as the value
    #write out results to file
    outfile_db.write("Query" + "\t" + "PFam_Hit" + "\t" + "E-Value" + "\t" + "Sum_Score" + "\t" + "Query_Coverage" + \
    "\t" + "eggNOG_Seed_Ortholog" + "\t" + "Seed_E-Value" + "\t" + "Seed_Score" + "\t" + "KEGG_Pathway" + \
    "\t" + "KEGG_Reaction" + "\t" + "PFams" + "\t" + "DeepLoc_Location_Prediction" + "\t" + \
    "Extracellular_Probability" + "\t" + "Cell_Membrane_Probability" + "\t" + \
    "SignalP_Prediction" + "\t" + "Probability" + "\n")
    #first, write out the column headers used in the file
    #then, write out the contents of the query_dict dictionary
    for species_key in query_dict:
        #iterate through the query_dict dictionary using its keys
        outfile_db.write(str(species_key) + "\t" + str(query_dict[species_key][0]) + "\t" + \
    str(query_dict[species_key][1]) + "\t" + str(query_dict[species_key][2]) + "\t" + \
    str(query_dict[species_key][3]) + "\t" + str(query_dict[species_key][4]) + "\t" + \
    str(query_dict[species_key][5]) + "\t" + str(query_dict[species_key][6]) + "\t" + \
    str(query_dict[species_key][7]) + "\t" + str(query_dict[species_key][8]) + "\t" + \
    str(query_dict[species_key][9]) + "\t" + str(query_dict[species_key][10]) + "\t" + \
    str(query_dict[species_key][11]) + "\t" + str(query_dict[species_key][12]) + "\t" + \
    str(query_dict[species_key][13]) + "\t" + str(query_dict[species_key][13]) + "\n")
        #results are written out to the output file in tab-delimited format
```

```
#running the program from the command line
#model:
python combo_OG_results.py PFam_EN_Parsed EggNOG_Parsed DeepLoc_Parsed SignalP_Parsed Output_DB
#application:
python ..\combo_OG_results.py BM_anaeromoeba_EN_PFam.txt BM_anaeromoeba_EN_eggNOG.txt BM_anaeromoeba_DL_DeepLoc.txt \
  BM_newprots_may21_SignalP.txt BM_anaeromoeba_DB.txt
python ..\combo_OG_results.py BS_anaeromoeba_EN_PFam.txt BS_anaeromoeba_EN_eggNOG.txt BS_anaeromoeba_DL_DeepLoc.txt \
  BS_newprots_may21_SignalP.txt BS_anaeromoeba_DB.txt
python ..\combo_OG_results.py D_fragilis_EN_PFam.txt D_fragilis_EN_eggNOG.txt D_fragilis_DL_DeepLoc.txt \
  Dientamoeba_fragilis_SignalP.txt D_fragilis_DB.txt
python ..\combo_OG_results.py H_meleagridis_EN_PFam.txt H_meleagridis_EN_eggNOG.txt H_meleagridis_DL_DeepLoc.txt \
  Histomonas_meleagridis_SignalP.txt H_meleagridis_DB.txt
python ..\combo_OG_results.py P_hominis_EN_PFam.txt P_hominis_EN_eggNOG.txt P_hominis_DL_DeepLoc.txt \
  Pentatrichomonas_hominis_SignalP.txt P_hominis_DB.txt
python ..\combo_OG_results.py SC_anaeromoeba_EN_PFam.txt SC_anaeromoeba_EN_eggNOG.txt SC_anaeromoeba_DL_DeepLoc.txt \
  SC_newprots_may21_SignalP.txt SC_anaeromoeba_DB.txt
python ..\combo_OG_results.py T_foetus_EN_PFam.txt T_foetus_EN_eggNOG.txt T_foetus_DL_DeepLoc.txt \
  Trichomonas_foetus_SignalP.txt T_foetus_DB.txt
python ..\combo_OG_results.py T_gallinarum_EN_PFam.txt T_gallinarum_EN_eggNOG.txt T_gallinarum_DL_DeepLoc.txt \
  Tetratrichomonas_gallinarum_SignalP.txt T_gallinarum_DB.txt
python ..\combo_OG_results.py T_vaginalis_GB_EN_PFam.txt T_vaginalis_GB_EN_eggNOG.txt T_vaginalis_GB_DL_DeepLoc.txt \
  Trichomonas_vaginalis_GenBank_SignalP.txt T_vaginalis_GB_DB.txt
python ..\combo_OG_results.py T_vaginalis_RS_EN_PFam.txt T_vaginalis_RS_EN_eggNOG.txt T_vaginalis_RS_DL_DeepLoc.txt \
  Trichomonas_vaginalis_RefSeq_SignalP.txt T_vaginalis_RS_DB.txt
#done
```

Script adding in the data from OrthoFinder (saved in og_DB_plusOF.py):

- Check if there isn't a version of Orthogroups.tsv that has the proteins on the side (but I doubt it)
    - Might need to write parser that reorganizes the data like that: 1st column with protein name, and second column with associated OGs
    - Or can just do list comprehension: import the columns as lists, and iterate through them
    - But that might be harder to code, vs. a script that reorganizes the file separate from the script that adds the data to the larger database files
- If protein is in the 2 parsed files, add the OGs to a new column
    - Total 2 new columns here: OGs that exist in all species, & OGs unique to *T. vaginalis*
- General idea: iterating through, column by column. For each cell of each column, I extract the contents and .split() using the commas as the basis, and compile it into a list. Then immediately use those protein ids to create the start of a new dictionary, where the key is the query protein, and the value is a list that I'll append to as I go through. If I hit a protein I've already encountered (or the program does), I append that OG ID to the dictionary value list. Then just make a new dataframe from the dictionary.
- The script for further parsing of the *T. vaginalis* OGs is saved in og_DB_plusOF_TvagU.py

```
#!/bin/python
"""

Title: og_DB_plusOF.py
Date: 10.08.2021
Author: Virág Varga

Description:
    This program takes as input a flat database output by combo_OG_results.py for
        one species, along with a filtered or unfiltered results file of the
        OrthoFinder program (ie. Orthogroups.tsv in .csv format or a filtered version
        of this file maintaining formatting); and parses through these two files to output
        a new version of the species' orthologs database which includes orthologous
        groups identified by OrthoFinder.

List of functions:
    flatten (Source: https://stackoverflow.com/a/10824086)

List of standard and non-standard modules used:
    sys
    os
    pandas
    numpy
    re

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the combo_OG_results.py results file, as
        well as a .csv version of the OrthoFinder results file Orthogroups.tsv or a
        filtered derivative into dataframes.
    3. Defining function `flatten`.
    4. Setting up variables and objects used later in the script.
    5. Parsing through the OrthoFinder results-derived dataframe to populate a
        dictionary with protein ids as keys and assigned orthologous group ids as
        values.
    6. Iterating over the species database-derived dataframe and the orthologous
        group-containing dictionary, in order to import the orthologous group ids
        associated with various proteins into the species database.
    5. Writing out the results to a .csv file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of the flat database created for the species or
        strain by the combo_og_results.py program; as well as the Orthogroups.tsv results
        file from Orthofinder, or a filtered version of this file that has maintained
        the original's formatting. Note here that the input file derived from the
        OrthoFinder results must be in .csv format.
    - All input and output files are user-defined: This means the user must ensure that
        the correct file names have been assigned to the program. Note that the assigned
        results file must have a .csv extension.

Inputs and Outputs:
    - Inputs:
        - Species_DB: Tab-delimited flat database (.txt extension); the output of the
            combo_OG_results.py program.
        - OG_OF_infile: Comma-delimited results file (.csv extension) derived from
            the original Orthogroups.tsv results file produced by OrthoFinder. Filtered
            versions of the file will work, as long as the structure of the results file
            (OG IDs in the first column, proteins separated by commas in columns by species)
            is maintained.
    - Output: Species_DB_OF: Comma-delimited (.csv extension) flat database that will
        contain the contents of the species database, along with the orthologous groups
        assigned by the OrthoFinder program.

Usage
    ./og_DB_plusOF.py Species_DB OG_OF_infile Species_DB_OF
    OR
    python og_DB_plusOF.py Species_DB OG_OF_infile Species_DB_OF

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""
```

```
# Step 1: Importing necessary modules, assigning command line arguments

#import necessary modules
import sys #allows assignment of command line arguments
import os #grants access to computer files & paths
import pandas as pd #facilitates manipulation of dataframes in Python
import numpy as np #facilitates manipulation of arrays in Python
import re #enable regex pattern matching


#assign command line arguments: inputs & outputs
#input files
Species_DB = sys.argv[1]
#Species_DB = "BM_anaeromoeba_DB.txt"
OG_OF_infile = sys.argv[2]
#OG_OF_infile = "All_species_OGs.csv"
#output file
Species_DB_OF = sys.argv[3]
#Species_DB_OF = "BM_anaeromoeba_DB_OFall.csv"


# Step 2: Importing data from input files into Pandas dataframes

#the input species database
#read in the input species database file, assigning the first row as a header row
#`sep="\t"` tells Pandas that the file is tab-delimited
species_df = pd.read_csv(Species_DB, sep="\t", header=0)
#set the first column (containing the protein query ids) as an index
species_df.set_index('Query')

#the OrthoFinder assignments database
#read in the input csv file, assigning the first row as a header row
ortho_df = pd.read_csv(OG_OF_infile, header=0)
#replace empty cells with NaN
ortho_df = ortho_df.replace(r'^\s*$', np.NaN, regex=True)
#filter the contents of the database, and drop cells containing NaN to the bottom
ortho_df_null = ortho_df.apply(lambda x: pd.Series(x.dropna().values))
#replace cells containing NaN with '' (an empty string)
#this is necessary because NaN cells are interpreted as containing numerical data
#which will interfere with the parsing of the dataframe's values (cell contents) as strings
ortho_df_null = ortho_df_null.fillna('')
#the ortho_df dataframe maintains the relationships between protein ids and orthologous groups
#it will be used to extract the orthologous group assignments into the dictionary
#the ortho_df_null dataframe contains no 'Nan's, and so can be parsed to extract the protein ids


# Step 3: Defining function `flatten`

#key function to flatten lists
#ie. extract the contents of lists nested within other lists into the larger list
#this simplifies the contents of the original list back to containing only strings
#Source: https://stackoverflow.com/a/10824086
def flatten(items, seqtypes=(list, tuple)):
    #the function accepts either a list or a tuple as input
    for i, x in enumerate(items):
        #iterate through the elements of the list or tuple (contained in x)
        #while keeping track of the index (contained in i)
        while i < len(items) and isinstance(items[i], seqtypes):
            #while the index remains less than the length of the list/tuple
            items[i:i+1] = items[i]
            #the contents of the nested list/tuple are extracted into the larger one
    return items #the flattened list/tuple is returned at the end


# Step 4: Setting up variables & objects to be used later

#creating column header for new column based on OrthoFinder results file used as input
#extract the file name
base = os.path.basename(OG_OF_infile)
#extract the base file name (ie. without the full path)
name_base = os.path.splitext(base)[0]
#extract the base of the file name (ie. without the extension)
```

```
column_name = name_base.split(".")[0]
#this will be used as the column header for the new column to be added to the database
#next, create an empty column to populate
species_df[column_name] = "None"
#note that new columns are automatically added to the end of the dataframe, unless otherwise specified
#'None' is used here because empty cells run through Pandas write out to 'None' in the output csv file
#so filling the last column with 'None' values will make it match the rest of the spreadsheet

#create empty dictionary
og_dict = {}
#keys will be protein query ids; values will be the assigned orthologous groups


# Step 5: Parsing through the OrthoFinder results to create a dictionary
# which links protein ids with orthologous groups

#parse through the OrthoFinder results to extract the protein-orthologous group pairings
for label, content in ortho_df_null.iteritems():
    #iterate through the ortho_df_null dataframe on the basis of column headers and column contents
    #label here refers to the column headers; content refers to the content of the cells in a columns
    series_prots = []
    #empy list series_prots will be filled with the proteins assigned to any given species
    #ie. in any given column
    if label == 'Orthogroup':
        #the first column (containing the orthologous group assignments) is skipped
        pass
    else:
        #the columns containing proteins ids (each column associated with a species) are parsed
        #the manipulations done in this loop are executed column by column
        series_prots = list(content)
        #the contents of the column are saved to the series_prots list
        for c, entry in enumerate(series_prots):
            #iterate over the contents of list series_prots
            #entry contains the item in the list; c contains the index of that item
            if ',' in entry:
                #any cell (value in Pandas dataframe-specific jargon) can contain multiple proteins
                #when there are multiple proteins, these are separated by a comma (',') and a space (' ')
                #the way `list(content)` imports the values into a list, each cell becomes 1 string in the list
                #so in places where a cell contained multiple protein ids,
                #they must be identified and manually separated
                new_entry = entry.split(",")
                #`split(',')` here separates the long string containing several protein ids at the commas (',')
                #into a list of smaller strings, each 1 protein id
                series_prots[c] = new_entry
                #this list of protein ids replaces the original string at index c
        flatten(series_prots)
        #function `flatten` is called to flatten out the nested lists, extracting their contents to the main list
        #now instead of a list containing both strings and lists, list series_prots contains only strings
        #some of the list entries have a ' ' character in them
        #these spaces remain from the protein id lists within the cells, and need to be deleted
        for s, member in enumerate(series_prots):
            #iterate over the contents of list series_prots
            #member contains the item in the list; s contains the index of that item
            if ' ' in member:
                #identify items in the list that contain a space (' ')
                new_member = re.sub(' ', '', member)
                #remove the space (' ') from the string
                series_prots[s] = new_member
                #replace the original string with the new string that no longer has the space (' ')
        #remove empty strings if they exist, using list comprehension
        series_prots_new = [x for x in series_prots if x]
        #empty strings may have been filtered out before automatically,
        #but this does it explicitly, just in case
        #now we can search for the indices of the protein ids that have been extracted
        #in order to populate the protein id-orthologous group assignment dictionary
        #note that doubles of proteins are not a concern
        #OrthoFinder only assigns a protein to 1 orthologous groups
        for i in series_prots_new:
            #iterate through the list of protein ids
            if i in ortho_df.values:
                #iterate through the contents of the columns in the ortho_df dataframe
                #this dataframe mirrors the originally imported file
                #and thus maintains the link between protein ids and orthologous group assignments
                query_loc = np.where(ortho_df.values == i)
                #use numpy to locate the cell which contains the protein id
                #and save the location information to variable query_loc
                #numpy will output these results in its own formatting, so need to extract the indexing values
                query_loc = [x[0] for x in query_loc]
                #list comprehension can be used to extract the row index
                query_og = ortho_df.iloc[query_loc[0]]['Orthogroup']
                #the row index of the protein id is used in conjunction with
                #the selection of the 'Orthogroup' column to extract
                #the orthologous group assignment of the protein
                og_dict[i] = query_og
                #the og_dict dictionary is populated using the protein id as the key
                #and its assigned orthologous group as the associated value


# Step 6: Adding the orthologous group information to the species_df dataframe

#iterate through the species_df dataframe
#where the protein query id matches a key in the og_dict dictionary, fill in the empty cell (value)
#in the final column with the corresponding dictionary value (ie. orthologous group ID)
for id_num, prot_id in enumerate(species_df['Query']):
    #iterate over the contents of the first column of the species_df dataframe (header/column name  = 'Query')
    #prot_id contains the value in the cell; id_num contains the index of that value
    for og_key in og_dict:
        #iterate through the og_dict dictionary using its keys
        if prot_id == og_key:
            #if the protein id in the species_df dataframe matches the key in the og_dict dictionary
            #extract the associated value from dictionary og_dict (ie. orthologous group assignment)
            #and place it in the appropriately indexed cell
            species_df.iat[id_num, -1] = og_dict[og_key]


# Step 7: Write out new species database to a csv file

#write the resulting new dataframe out to the assigned result file
species_df.to_csv(Species_DB_OF, index=False)
#use `index=False` to prevent Pandas from inserting a column at the front of the file containing index values
```

```
#running the program
#run the All_species_OGs.csv file against all of the species & strains
#model:
python og_DB_plusOF.py Species_DB OG_OF_infile Species_DB_OF
#adapting it:
python ..\og_DB_plusOF.py BM_anaeromoeba_DB.txt ..\All_species_OGs.csv BM_anaeromoeba_DB_OFall.csv
python ..\og_DB_plusOF.py BS_anaeromoeba_DB.txt ..\All_species_OGs.csv BS_anaeromoeba_DB_OFall.csv
python ..\og_DB_plusOF.py D_fragilis_DB.txt ..\All_species_OGs.csv D_fragilis_DB_OFall.csv
python ..\og_DB_plusOF.py H_meleagridis_DB.txt ..\All_species_OGs.csv H_meleagridis_DB_OFall.csv
python ..\og_DB_plusOF.py P_hominis_DB.txt ..\All_species_OGs.csv P_hominis_DB_OFall.csv
python ..\og_DB_plusOF.py SC_anaeromoeba_DB.txt ..\All_species_OGs.csv SC_anaeromoeba_DB_OFall.csv
python ..\og_DB_plusOF.py T_foetus_DB.txt ..\All_species_OGs.csv T_foetus_DB_OFall.csv
python ..\og_DB_plusOF.py T_gallinarum_DB.txt ..\All_species_OGs.csv T_gallinarum_DB_OFall.csv
python ..\og_DB_plusOF.py T_vaginalis_GB_DB.txt ..\All_species_OGs.csv T_vaginalis_GB_DB_OFall.csv
python ..\og_DB_plusOF.py T_vaginalis_RS_DB.txt ..\All_species_OGs.csv T_vaginalis_RS_DB_OFall.csv
#then run the Tvag_unique_OGs.csv file against the 2 T. vaginalis files
#made slightly modified version of the og_DB_plusOF.py script for this
#since the input database is a .csv instead of a tab-delimited .txt file
#new script saved under og_DB_plusOF_TvagU.py
python ..\og_DB_plusOF_TvagU.py T_vaginalis_GB_DB_OFall.csv ..\Tvag_unique_OGs.csv T_vaginalis_GB_DB_OFall_TvagU.csv
python ..\og_DB_plusOF_TvagU.py T_vaginalis_RS_DB_OFall.csv ..\Tvag_unique_OGs.csv T_vaginalis_RS_DB_OFall_TvagU.csv
#done!
```

Script to score the OG results (saved in score_OG_DB.py):

- scores the results
- outputs a filtered and unfiltered scored database

```
#!/bin/python
"""
```

```
Title: score_OG_DB.py
Date: 16.08.2021
Author: Virág Varga

Description:
    This program scores the predictions generated for the protein queries present
        in a species database generated by either combo_OG_results.py, og_DB_plusOF.py
        or og_DB_plusOF_TvagU.py. The scores are then used to generate a filtered version
        of the existing dataframe.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas
    numpy

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas to import the contents of the input database, and creating a
        new column in the dataframe for the results of the scoring.
    3. Scoring the predictions generated for each protein query sequence in the file,
        and addin the results to the dataframe.
    4. Minor aesthetic changes to the database for the sake of consistency.
    5. Generating a new filtered dataframe based on the results of the scoring process.
    6. Writing out the results to two csv files: one containing all of the protein
        queries with their associated scores, and one containing only the contents of
        the filtered dataframe.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a flat database created by either combo_OG_results.py,
        og_DB_plusOF.py or og_DB_plusOF_TvagU.py. If following the Trich_Parab official
        workflow, the database produced by either og_DB_plusOF.py or og_DB_plusOF_TvagU.py
        should be used as input.
    - Only the input file is user-defined; the naming of the output file is automatic.

Usage
    ./score_OG_DB.py input_db
    OR
    python score_OG_DB.py input_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python
import numpy as np #facilitates manipulation of arrays in Python


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "BM_anaeromoeba\BM_anaeromoeba_DB_OFall.csv"
#output_db and filtered_db names are based on the input_db name
#output_db will contain a copy of the original database, plus scores
output_db = ".".join(input_db.split('.')[:-1]) + '__score.csv'
#filtered_db will contain a filtered version of the scored database
filtered_db = ".".join(input_db.split('.')[:-1]) + '__scoreGood.csv'


#read in the input OG database file, assigning the first row as a header row
ortho_df = pd.read_csv(input_db, header=0)
#set the first column (containing the protein query ids) as an index
ortho_df.set_index('Query')

#add new column to the database, for the scoring
ortho_df['Score'] = np.nan
#using 'NaN' here instead of 'None' because 'NaN' is numeric
```

```python
#now iterate through the ortho_df dataframe, and score each protein based on the DeepLoc & SignalP results
for index, row in ortho_df.iterrows():
    #iterate through the dataframe row by row
    score = 0
    #create a counter for the score
    #this will be rewritten to 0 with every iteration of the loop
    dl_pred = row['DeepLoc_Location_Prediction']
    #save the DeepLoc prediction to variable dl_pred
    sp_pred = row['SignalP_Prediction']
    #save the SignalP prediction to variable sp_pred
    if dl_pred == 'Extracellular':
        #identify proteins DeepLoc predicts to be extracellular
        dl_prob = row['Extracellular_Probability']
        #save the probability that the protein is predicted to be extracellular to variable dl_prob
        if dl_prob >= 0.5:
            #if dl_prob is greater than or equal to 0.5 (>=0.5),
            #add 1 point to the score
            score += 1
        else:
            #if dl_prob is less than 0.5,
            #add 0.5 to the score
            score += 0.5
    if dl_pred == 'Cell_membrane':
        #identify proteins DeepLoc predicts to be targeted to the cell membrane
        dl_cm_prob = row['Cell_Membrane_Probability']
        #save the probability that the protein is targeted to the cell membrane to variable dl_cm_prob
        if dl_cm_prob >= 0.5:
            #if dl_cm_prob is greater than or equal to 0.5 (>=0.5),
            #add 1 point to the score
            score += 1
    else:
            #if dl_cm_prob is less than 0.5,
            #add 0.5 to the score
            score += 0.5
    if sp_pred == 'SP(Sec/SPI)':
        #identify proteins SignalP identifies to be secreted
        sp_prob = row['Probability']
        #save te probability that the protein will be secreted to variable sp_prob
        if sp_prob >= 0.5:
            #if sp_prob is greater than or equal to 0.5 (>=0.5),
            #add 1 point to the score
            score +=1
        else:
            #if sp_prob is less than 0.5,
            #add 0.5 to the score
            score += 0.5
    #fill in the calculated score for the protein in the appropriate location in the dataframe
    ortho_df.at[index, 'Score'] = score


#minor editing for aesthetic reasons
ortho_df.replace("None", "-", inplace=True)
#some of the imported data uses '-' if there is no result
#but if there was nothing there, 'None' is printed by Pandas
#this formatting fix makes the formatting the same throughout the database


#create new filtered datafarme based on the scores
filt_ortho_df = ortho_df[ortho_df['Score'] >= 1]
#filt_ortho_df = ortho_df[ortho_df.Score >= 1] #alternate version
#only scores >=1 will be kept in this new dataframe


#write the scored dataframe out to the assigned result file
with open(output_db, "w"):
    #open the output_db file for writing
    ortho_df.to_csv(output_db, index=False)
    #reulsts will be written out to a csv file
    #use `index=False` to prevent Pandas from inserting a column at the front of the file containing index values
#next write out the filtered dataframe to its assigned outfile
with open(filtered_db, "w"):
    #open the filtered_db file for writing
    filt_ortho_df.to_csv(filtered_db, index=False)
    #results will be written out to a csv file
```

Running the script:

```
#model:
python score_OG_DB.py input_db
#adapting it:
#run each of these from within the respective species directory
#within the DatabaseCompilation/ directory
python ..\score_OG_DB.py BM_anaeromoeba_DB_OFall.csv
python ..\score_OG_DB.py BS_anaeromoeba_DB_OFall.csv
python ..\score_OG_DB.py D_fragilis_DB_OFall.csv
python ..\score_OG_DB.py H_meleagridis_DB_OFall.csv
python ..\score_OG_DB.py P_hominis_DB_OFall.csv
python ..\score_OG_DB.py SC_anaeromoeba_DB_OFall.csv
python ..\score_OG_DB.py T_foetus_DB_OFall.csv
python ..\score_OG_DB.py T_gallinarum_DB_OFall.csv
#use the *TvagU.csv files for the scoring for T. vaginalis versions
python ..\score_OG_DB.py T_vaginalis_GB_DB_OFall_TvagU.csv
python ..\score_OG_DB.py T_vaginalis_RS_DB_OFall_TvagU.csv
#done
```

Create one giant database concatenating the files - 2 steps:

- add in column (column 2; pythonic index 1) containing species/strain ID (add_speciesID_DB.py)
- concatenate all of the files (do this with awk code included below)

```
#!/bin/python
"""

Title: add_speciesID_DB.py
Date: 16.08.2021
Author: Virág Varga

Description:
    This program adds a second column into a csv file containing the string input
        by the user. The intention is to add a species designation to a database
        containing the protein queries from 1 species, prior to concatenation of
        multiple similar files.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas to import the contents of the input database, and addinging a
        new column in the dataframe for the species designation.
    3. Writing out the results to two csv files: one containing all of the protein
        queries with their associated scores, and one containing only the contents of
        the filtered dataframe.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - The input species designation is user-defined; therefor the user must ensure
        that the designation is correct.

Usage
    ./add_speciesID_DB.py input_db species_name
    OR
    python add_speciesID_DB.py input_db species_name
```

```
This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "BM_anaeromoeba\BM_anaeromoeba_DB_OFall__scoreGood.csv"
species_name = sys.argv[2]
#species_name = "BM_anaeromoeba"
#output_db name is based on the input_db name
output_db = ".".join(input_db.split('.')[:-1]) + '_sp.csv'


#read in the input OG database file, assigning the first row as a header row
ortho_df = pd.read_csv(input_db, header=0)
#set the first column (containing the protein query ids) as an index
ortho_df.set_index('Query')


#add new column to the database, for the species designation
ortho_df.insert(loc=1, column='Species_ID', value=species_name)


#write the scored dataframe out to the assigned result file
with open(output_db, "w"):
    #open the output_db file for writing
    ortho_df.to_csv(output_db, index=False)
    #reulsts will be written out to a csv file
    #use `index=False` to prevent Pandas from inserting a column at the front of the file containing index values
```

```
#run the script above on all files, like so:
#model:
python add_speciesID_DB.py input_db species_name
#adapting it:
python ..\add_speciesID_DB.py BM_anaeromoeba_DB_OFall__scoreGood.csv BM_anaeromoeba
python ..\add_speciesID_DB.py BS_anaeromoeba_DB_OFall__scoreGood.csv BS_anaeromoeba
python ..\add_speciesID_DB.py D_fragilis_DB_OFall__scoreGood.csv D_fragilis
python ..\add_speciesID_DB.py H_meleagridis_DB_OFall__scoreGood.csv H_meleagridis
python ..\add_speciesID_DB.py P_hominis_DB_OFall__scoreGood.csv P_hominis
python ..\add_speciesID_DB.py SC_anaeromoeba_DB_OFall__scoreGood.csv SC_anaeromoeba
python ..\add_speciesID_DB.py T_foetus_DB_OFall__scoreGood.csv T_foetus
python ..\add_speciesID_DB.py T_gallinarum_DB_OFall__scoreGood.csv T_gallinarum
python ..\add_speciesID_DB.py T_vaginalis_GB_DB_OFall_TvagU__scoreGood.csv T_vaginalis_GB
python ..\add_speciesID_DB.py T_vaginalis_RS_DB_OFall_TvagU__scoreGood.csv T_vaginalis_RS
#let's also do this for the non-filtered scored files
python ..\add_speciesID_DB.py BM_anaeromoeba_DB_OFall__score.csv BM_anaeromoeba
python ..\add_speciesID_DB.py BS_anaeromoeba_DB_OFall__score.csv BS_anaeromoeba
python ..\add_speciesID_DB.py D_fragilis_DB_OFall__score.csv D_fragilis
python ..\add_speciesID_DB.py H_meleagridis_DB_OFall__score.csv H_meleagridis
python ..\add_speciesID_DB.py P_hominis_DB_OFall__score.csv P_hominis
python ..\add_speciesID_DB.py SC_anaeromoeba_DB_OFall__score.csv SC_anaeromoeba
python ..\add_speciesID_DB.py T_foetus_DB_OFall__score.csv T_foetus
python ..\add_speciesID_DB.py T_gallinarum_DB_OFall__score.csv T_gallinarum
python ..\add_speciesID_DB.py T_vaginalis_GB_DB_OFall_TvagU__score.csv T_vaginalis_GB
python ..\add_speciesID_DB.py T_vaginalis_RS_DB_OFall_TvagU__score.csv T_vaginalis_RS
```

I wrote another python script to do the concatenation manually (saved to concat_OD_DBs.py).

```
#!/bin/python
"""

Title: concat_OG_DBs.py
```

```
Date: 10.08.2021
Author: Virág Varga

Description:
    This program concatenates the databases created after teh OG comparisons, accounting
        for the fact that the T. vaginalis databases have an extra column in them.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the separate species databases.
    3. Creating a combined dataframe of the non-T. vaginalis species dataframes,
        and add an empty column to them to align withe the proteins unique to
        T. vaginalis.
    4. Add the T. vaginalis dataframes to the larger dataframe.
    5. Writing out the results to a csv file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of the scored database files from the Trich_Parab
        project.
    - All input and output files are program-defined: only the option of using the
        filtered scored databases or non-filtered scored databases is presented as an
        option for the user.

Usage
    ./concat_OG_DBs.py input_option
    OR
    python concat_OG_DBs.py input_option

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows execution of code options from the command line
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments: input & output files
#working from the DatabaseCompilation/ directory
if sys.argv[1] == "scoreGood":
    #use when concatenating the filtered databases
    BM_anaero_db = "BM_anaeromoeba\BM_anaeromoeba_DB_OFall__scoreGood_sp.csv"
    BS_anaero_db = "BS_anaeromoeba\BS_anaeromoeba_DB_OFall__scoreGood_sp.csv"
    D_fragilis_db = "D_fragilis\D_fragilis_DB_OFall__scoreGood_sp.csv"
    H_meleagridis_db = "H_meleagridis\H_meleagridis_DB_OFall__scoreGood_sp.csv"
    P_hominis_db = "P_hominis\P_hominis_DB_OFall__scoreGood_sp.csv"
    SC_anaero_db = "SC_anaeromoeba\SC_anaeromoeba_DB_OFall__scoreGood_sp.csv"
    T_foetus_db = "T_foetus\T_foetus_DB_OFall__scoreGood_sp.csv"
    T_gallinarum_db = "T_gallinarum\T_gallinarum_DB_OFall__scoreGood_sp.csv"
    #T. vaginalis files
    T_vaginalis_GB_db = "T_vaginalis_GB\T_vaginalis_GB_DB_OFall_TvagU__scoreGood_sp.csv"
    T_vaginalis_RS_db = "T_vaginalis_RS\T_vaginalis_RS_DB_OFall_TvagU__scoreGood_sp.csv"
    output_db = "All_species_OG_DB_scoreGood.csv"
elif sys.argv[1] == "score":
    #use when concatenating the non-filtered databases
    BM_anaero_db = "BM_anaeromoeba\BM_anaeromoeba_DB_OFall__score_sp.csv"
    BS_anaero_db = "BS_anaeromoeba\BS_anaeromoeba_DB_OFall__score_sp.csv"
    D_fragilis_db = "D_fragilis\D_fragilis_DB_OFall__score_sp.csv"
    H_meleagridis_db = "H_meleagridis\H_meleagridis_DB_OFall__score_sp.csv"
    P_hominis_db = "P_hominis\P_hominis_DB_OFall__score_sp.csv"
    SC_anaero_db = "SC_anaeromoeba\SC_anaeromoeba_DB_OFall__score_sp.csv"
    T_foetus_db = "T_foetus\T_foetus_DB_OFall__score_sp.csv"
    T_gallinarum_db = "T_gallinarum\T_gallinarum_DB_OFall__score_sp.csv"
    #T. vaginalis files
    T_vaginalis_GB_db = "T_vaginalis_GB\T_vaginalis_GB_DB_OFall_TvagU__score_sp.csv"
    T_vaginalis_RS_db = "T_vaginalis_RS\T_vaginalis_RS_DB_OFall_TvagU__score_sp.csv"
    output_db = "All_species_OG_DB_scoreAll.csv"
else:
    print("Please use an applicable option: Either scoreGood or score.")


#importing species data into pandas dataframes
#first, the "regular" species dataframes
#read in the input OG database file, assigning the first row as a header row
BM_anaero_df = pd.read_csv(BM_anaero_db, header=0)
#set the first column (containing the protein query ids) as an index
BM_anaero_df.set_index('Query')
#repeat this process for all species
BS_anaero_df = pd.read_csv(BS_anaero_db, header=0)
BS_anaero_df.set_index('Query')
D_fragilis_df = pd.read_csv(D_fragilis_db, header=0)
D_fragilis_df.set_index('Query')
H_meleagridis_df = pd.read_csv(H_meleagridis_db, header=0)
H_meleagridis_df.set_index('Query')
P_hominis_df = pd.read_csv(P_hominis_db, header=0)
P_hominis_df.set_index('Query')
SC_anaero_df = pd.read_csv(SC_anaero_db, header=0)
SC_anaero_df.set_index('Query')
T_foetus_df = pd.read_csv(T_foetus_db, header=0)
T_foetus_df.set_index('Query')
T_gallinarum_df = pd.read_csv(T_gallinarum_db, header=0)
T_gallinarum_df.set_index('Query')
#next, the T. vaginalis dataframes
T_vaginalis_GB_df = pd.read_csv(T_vaginalis_GB_db, header=0)
T_vaginalis_GB_df.set_index('Query')
T_vaginalis_RS_df = pd.read_csv(T_vaginalis_RS_db, header=0)
T_vaginalis_RS_df.set_index('Query')


#concatenating non-T. vaginalis files
frames_1 = [BM_anaero_df, BS_anaero_df, D_fragilis_df, H_meleagridis_df, P_hominis_df, SC_anaero_df, T_foetus_df, \
  T_gallinarum_df]
#put the dataframes to be concatenated into a list
temp_df = pd.concat(frames_1)
#concatenate the dataframes into a new dataframe


#adding empty column full of "-" as second to last column
#add new column to the database, for the species designation
temp_df.insert(loc=18, column='Tvag_unique_OGs', value="-")


#adding the T. vaginalis dataframes to the large concatenated dataframe
#creating a list of the dataframes to be concatenated
frames_2 = [temp_df, T_vaginalis_GB_df, T_vaginalis_RS_df]
#concatenating the dataframes
final_df = pd.concat(frames_2)


#write the scored dataframe out to the assigned result file
with open(output_db, "w"):
    #open the output_db file for writing
    final_df.to_csv(output_db, index=False)
    #results will be written out to a csv file
    #use `index=False` to prevent Pandas from inserting a column at the front of the file containing index values
```

Running it:

```
#from the Trich_Parab/Analyses/DatabaseCompilation/ directory
#model:
python concat_OG_DBs.py input_option
#adapting it:
python concat_OG_DBs.py scoreGood
python concat_OG_DBs.py score
```

Modified decoder script for protein names (saved in decodeHeaders.py):

- Based on/modification of reverseHeaders.py

```
#!/bin/python
"""

Title: decodeHeaders.py
Date: 28.07.2021
Author: Virág Varga

Description:
    This program decodes the random alphanumeric headers assigned by the
        assignFASTAheaders.py program with the original FASTA headers, using the
        reference file created by the other program as a guide. It is intended
        for use on the *_OGall*.csv species & strain databases.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    re
    pandas

Procedure:
    1. Loading required modules & assigning command line arguments.
    2. Using Pandas to load the contents of the reference file into a dictionary.
    3. Parsing the input database file in order to match the random headers to the
        original headers via the reference file.
    4. Writing out the new database file with the original FASTA headers.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - The user needs to check that the correct reference file is assigned.

Usage
    ./decodeHeaders.py input_db ref_doc
    OR
    python decodeHeaders.py input_db ref_doc

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""

#import necessary modules
import sys #allows execution of script from command line
import re #enables regex pattern matching
import pandas as pd #allow manipulation of data files


#load input and output files
input_db = sys.argv[1]
#input_db = "BM_anaeromoeba_DB_OFall.csv"
ref_doc = sys.argv[2]
#ref_doc = "encoding_summary_ref.txt"
#output_db name is based on the input_fasta name
output_db = ".".join(input_db.split('.')[:-1]) + '__decode.csv'


#create and populate dictionary using the reference file
with open(ref_doc, "r") as inref:
    REF = pd.read_csv(inref, sep="\t", header=None)
    REF = REF.set_index(0)
    ref_dict = REF.T.to_dict('list')
    #note that this manner of conversion makes the dictionary value a list

#reading in the input database into a Pandas dataframe
species_df = pd.read_csv(input_db, header=0)
#set the first column (containing the protein query ids) as an index
species_df.set_index('Query')
```

```
#replace the spaces (' ') in the FASTA headers with underscores ('_')
for key, value in ref_dict.items():
    #iterate through the ref_dict dictionary
    if ' ' in value[0]:
        #identify values which contain spaces (' ')
        value[0] = re.sub(' ', '_', value[0])
        #replace spaces in the values with underscores ('_')


#iterate through the species_df dataframe
#where the protein query id matches a key in the ref_dict dictionary, fill in the empty cell (value)
#in the final column with the corresponding dictionary value (ie. original FASTA header)
for id_num, prot_id in enumerate(species_df['Query']):
    #iterate over the contents of the first column of the species_df dataframe (header/column name  = 'Query')
    #prot_id contains the value in the cell; id_num contains the index of that value
    for ref_key in ref_dict:
        #iterate through the og_dict dictionary using its keys
        if prot_id == ref_key:
            #if the protein id in the species_df dataframe matches the key in the og_dict dictionary
            #extract the associated value from dictionary og_dict (ie. orthologous group assignment)
            #and place it in the appropriately indexed cell
            species_df.iat[id_num, 0] = ref_dict[ref_key][0]
            #indexing the ref_dict value (FASTA header) makes the contents print as a string instead of a list


#write out the results to a new .csv file
species_df.to_csv(output_db, index=False)
```

```
#running the program
#model:
python decodeHeaders.py input_db ref_doc
#adapting it:
python ..\decodeHeaders.py BM_anaeromoeba_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py BS_anaeromoeba_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py D_fragilis_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py H_meleagridis_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py P_hominis_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py SC_anaeromoeba_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_foetus_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_gallinarum_DB_OFall__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_vaginalis_GB_DB_OFall_TvagU__scoreGood.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_vaginalis_RS_DB_OFall_TvagU__scoreGood.csv ..\encoding_summary_ref.txt
#and the complete database
python decodeHeaders.py All_Species_OG_DB.csv encoding_summary_ref.txt
#done!
###
#new version because I don't want to do all of that
python decodeHeaders.py .\*\*_sp.csv encoding_summary_ref.txt
#well that didn't work so the original way it is
python ..\decodeHeaders.py BM_anaeromoeba_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py BM_anaeromoeba_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py BS_anaeromoeba_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py BS_anaeromoeba_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py D_fragilis_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py D_fragilis_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py H_meleagridis_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py H_meleagridis_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py P_hominis_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py P_hominis_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py SC_anaeromoeba_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py SC_anaeromoeba_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_foetus_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_foetus_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_gallinarum_DB_OFall__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_gallinarum_DB_OFall__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_vaginalis_GB_DB_OFall_TvagU__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_vaginalis_GB_DB_OFall_TvagU__score_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_vaginalis_RS_DB_OFall_TvagU__scoreGood_sp.csv ..\encoding_summary_ref.txt
python ..\decodeHeaders.py T_vaginalis_RS_DB_OFall_TvagU__score_sp.csv ..\encoding_summary_ref.txt
#the complete databases
```

```
python decodeHeaders.py All_species_OG_DB_scoreAll.csv encoding_summary_ref.txt
python decodeHeaders.py All_species_OG_DB_scoreGood.csv encoding_summary_ref.txt
```

Create pivot table flipping the OGs and protein query IDs from the original Orthogroups.tsv file (code saved in pivot_OGs.py).

```
#!/bin/python
"""

Title: pivot_OGs.py
Date: 18.08.2021
Author: Virág Varga

Description:
    This program pivots the Orthogroups.tsv file's data, creating an output file that
        contains query proteins in individual columns and the OG associated with them
        in a cell in the same row.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas
    itertools

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas to import the contents of the Orthogroups.tsv file into a
        dataframe.
    3. Condensing the contents of all species columns into 1 large column, then
        separating out the protein ids into unique cells (instead of comma-separated
        strings in each cell), and then flip the column order so that the query
        protein IDs are in the first column.
    4. Writing out the results to a csv file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of the OrthoFinder results file, Orthogroups.tsv;
        or a file with identical formatting.

Usage
    ./pivot_OGs.py input_db
    OR
    python pivot_OGs.py input_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python
from  itertools import product


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "Orthogroups.tsv"
#output_db will contain the pivot table containing the OGs & associated PFams
output_db = sys.argv[2]
#output_db = "Orthogroups_pivot.csv"


#import input database into a Pandas dataframe
ortho_df = pd.read_csv(input_db, sep = '\t', header = 0)
#ortho_df = pd.read_csv(input_db, header = 0)


#first, condense the contents of the species columns,
#since the species information is unnecessary (already in large database)
ortho_df = ortho_df.melt(id_vars="Orthogroup", var_name="Species", value_name="Queries")
#melting the database like this creates 3 columns
#'Orthogroup' stays as is; the species headers go in column 2 'Species';
#and proteins go in column 3 'Queries'
#ortho_df.drop('Species', axis=1, inplace=True)
#originally inteded to drop the species, but decided against it in the end
#there may be identified proteins in the OrthoFinder OG search that didn't show up in the other program searches
#I won't use those proteins (since they won't have matches in the database)
#but best to keep them around as a reference


#pull apart the parts of the database that are comma-separated
#ie. flatten the database: each protein gets its own cell
#ref: https://stackoverflow.com/questions/50789834/parse-a-dataframe-column-by-comma-and-pivot-python
df1 = ortho_df.applymap(lambda x: x.split(', ') if isinstance (x, str) else [x])
#split apart the comma-separated lists of protein ids in each cell
df2 = pd.DataFrame([j for i in df1.values for j in product(*i)], columns=ortho_df.columns)
#give each protein ID its own cell, while still associated with the species ID and OG

#flip the column order, putting the proteins in the first column
final_df = df2.iloc[:, ::-1]


'''
###sanity check
test_df = final_df[final_df.duplicated(keep=False)]
test_df.duplicated(subset=['Queries'], keep='first')
test_list = test_df['Queries'].to_list()
test_list = list(dict.fromkeys(test_list))
#we're good
# I was worried because the duplicate test returned positive
#but the "duplicates" are empty cells, "NaN"
'''


#now write out the file
with open(output_db, "w"):
    #open the output_db file for writing
    final_df.to_csv(output_db, index=False)
```

Add the data from the program above to the larger databases (code saved in add_all_OGs.py):

```
#!/bin/python
"""

Title: add_all_OGs.py
Date: 19.08.2021
Author: Virág Varga

Description:
    This program adds the OG assignments to all proteins in the larger Trich_Parab
        database, using the pivoted Orthogroups.tsv file as the basis for input.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas
    numpy

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the larger trich_parab database, as well
        as the contents of the pivoted Orthogroups.tsv file, into two separate
        dataframes.
    3. Creating a combined dataframe including the OG assignments of all proteins in
        the trich_parab database.
    4. Writing out the results to a csv file.
```

```
Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a scored database file from the Trich_Parab
        project, along with the pivoted Orthogroups.tsv file.
    - All input and output files are user-defined; the user must therefore ensure
        proper naming conventions are followed.

Usage
    ./add_all_OGs.py input_db og_prot_db output_db
    OR
    python add_all_OGs.py input_db og_prot_db output_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python
import numpy as np #allow manipulation of dataframe arrays


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "All_species_OG_DB_scoreGood.csv"
#input_db contains the large input species database
og_prot_db = sys.argv[2]
#og_prot_db = "Orthogroups_pivot.csv"
#og_prot_db contains the OG pivot table to be mapped onto the input_db database
#output_db will contain the original database plus the assigned OGs
output_db = sys.argv[3]
#output_db = "All_species_allOG_DB_scoreGood.csv"


#import input databases into Pandas dataframes
#read in teh trich_parab database
input_df = pd.read_csv(input_db, header = 0)
input_df.set_index('Query')
#read in the pivoted Orthogroups.tsv file
og_pivot_df = pd.read_csv(og_prot_db, header = 0)
#for the sake of consistency and ease of dataframe manipulation, rename the query protein column
og_pivot_df.rename(columns={'Queries':'Query'}, inplace=True)
#remove the unnecessary 'Species' column
og_pivot_df.drop('Species', axis=1, inplace=True)
og_pivot_df.set_index('Query')


#now, merge the dataframes based on index
output_df = input_df.merge(og_pivot_df, how='left')


#now some clean-up

#move the 'Score' column to the end
# get a list of columns
cols = list(output_df)
# move the column to head of list using index, pop and insert
cols.insert(20, cols.pop(cols.index('Score')))
# use .loc to reorder
output_df = output_df.loc[:, cols]

#finally, fil 'NaN' values with '-' to match rest of database
output_df = output_df.replace(np.nan, '-', regex=True)


#now write out the file
with open(output_db, "w"):
    #open the output_db file for writing
    output_df.to_csv(output_db, index=False)
```

Running it:

```
#from the DatabaseCompilation/ directory
#model:
python add_all_OGs.py input_db og_prot_db output_db
#adapting it:
python add_all_OGs.py All_species_OG_DB_scoreGood.csv Orthogroups_pivot.csv All_species_allOG_DB_scoreGood.csv
python add_all_OGs.py All_species_OG_DB_scoreAll.csv Orthogroups_pivot.csv All_species_allOG_DB_scoreAll.csv
python add_all_OGs.py All_species_OG_DB_scoreGood__decode.csv Orthogroups_pivot.csv \
  All_species_allOG_DB_scoreGood__decode.csv
python add_all_OGs.py All_species_OG_DB_scoreAll__decode.csv Orthogroups_pivot.csv \
  All_species_allOG_DB_scoreAll__decode.csv
```

Pivot table containing OGs and associated PFam domains (saved in og2PFam_pivot.py; an earlier version of this script is saved in og2PFam_pivot__Ver1.py):

```
#!/bin/python
"""

Title: og2PFam_pivot.py
Date: 19.08.2021
Author: Virág Varga

Description:
    This program uses a larger trich_parab database containing all PFam and OG
        information to create a pivot table linking all OGs to their associated
        PFam domains.

List of functions:
    concat (Source: IPRpivot.py, Courtney Stairs)

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the trich_parab database.
    3. Copying out the PFam and Orthogroup information into a new dataframe, and
        pivoting that dataframe.
    4. Creating a condensed version of the PFam listss in order to make the output
        database easier to read and understand.
    5. Writing out the results to two csv files.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a trich_parab database file including both
        PFam and OG assignments.
    - Only the input database file has a user-assigned name.

Usage
    ./og2PFam_pivot.py input_db
    OR
    python og2PFam_pivot.py input_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allow execution of code from the command line
import pandas as pd #facilitates manipulation of dataframes in Python


# make function for joining strings with ','
#Source: IPRpivot.py, Courtney Stairs
def concat(str):
    return ','.join(str)
```

```
#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "All_species_allOG_Parab_DB_scoreAll.csv"
#output_db will contain the pivot table containing the OGs & associated PFams
output_db = "OGs2PFams_all_CLEAN.csv"


#read in the input OG database file, assigning the first row as a header row
input_df = pd.read_csv(input_db, header=0)


#select the columns that will be used to create the pivot table
selected_columns = input_df[['PFam_Hit', 'Orthogroup']]
#copy those columns only into a new dataframe
pfam_og_RAW = selected_columns.copy()


#now pivot the table
pfam_og_WIP = pd.pivot_table(pfam_og_RAW, index = 'Orthogroup', aggfunc=concat)


#unfortunatley, the dataframe this produces is so large that Excel won't open it
#so need to find a way to condense it

#create a new column that will house the condensed version of the data
pfam_og_WIP['PFam_Dense'] = '-'
#fill the new column with a placeholder "-"

#iterate through the dataframe adding in the condensed information to the new column
for index, row in pfam_og_WIP.iterrows():
    #iterate through the pfam_og_WIP dataframe row by row
    pfam_string = str(row['PFam_Hit'])
    #take the content of the 'PFam_Hit' column and extract it
    pfam_list = pfam_string.split(",")
    #seperate out the PFams ints a list
    pfam_dict = {i:pfam_list.count(i) for i in pfam_list}
    #create a miniature dictionary containing the assigned PFams
    #and the number of times that specific PFam was assigned to that OG
    pfam_dict.pop('-', None)
    #remove keys (and associated values) in pfam_dict that are the null '-' string
    if pfam_dict:
        #identify empty dictionaries,
        #and only iterate through the dictionary if it is NOT empty
        new_pfam_string = str(pfam_dict)
        #transform the "dcitionary" into a string format
        new_pfam_string = new_pfam_string.replace('}', '').replace('{', '')
        #remove the curly braces remaining from the conversion from a dictionary to a string
        new_pfam_string = new_pfam_string.replace("'", "")
        #remove single quotes around strings (ie. PFam hit names) in the output
        new_pfam_string = new_pfam_string.replace(',', ';')
        #aesthetic changes to the dictionary formatting to make it better for my eyes
        pfam_og_WIP.at[index, 'PFam_Dense'] = new_pfam_string
        #populate the 'PFam_Dense' column with the condensed PFam information


#removing the raw data column and creating the final database
pfam_og_df = pfam_og_WIP.drop('PFam_Hit', 1)


#now write out the file
with open(output_db, "w"):
    #open the output_db file for writing
    pfam_og_df.to_csv(output_db, index=True)
```

OGs not present in anaeromoebids (not necessarily present in all parabasalids) (script saved at the end of extractOGs.py):

```
#import necessary modules
import pandas as pd
from  itertools import product
```

```
#define inputs and outputs
ortho_input = "Orthogroups.tsv"
#ortho_input = "Orthogroups_extract.tsv"
nonAnaero_output = "NonAnaero_OGs.csv"
#nonAnaero_output = "NonAnaero_OGs_extract.csv"


#read in the input tsv file, assigning the first row as a header row
ortho_df = pd.read_csv(ortho_input, sep='\t', header=0)
#set the first column (containing the OGs) as an index
ortho_df.set_index('Orthogroup')


#identify rows where the 3 anaeromoebids have no proteins for that OG
nonAnaero_df = ortho_df[ortho_df['BM_newprots_may21.anaeromoeba_edit'].isnull() &
  ortho_df['BS_newprots_may21.anaeromoeba_edit'].isnull() &
  ortho_df['SC_newprots_may21.anaeromoeba_edit'].isnull()]

#now pivot the table for ease of import into the large trich_parab database

#first, condense the contents of the species columns,
#since the species information is unnecessary (already in large database)
nonAnaero_df = nonAnaero_df.melt(id_vars="Orthogroup", var_name="Species", value_name="Query")
#melting the database like this creates 3 columns
#'Orthogroup' stays as is; the species headers go in column 2 'Species';
#and proteins go in column 3 'Query'

#pull apart the parts of the database that are comma-separated
#ie. flatten the database: each protein gets its own cell
#ref: https://stackoverflow.com/questions/50789834/parse-a-dataframe-column-by-comma-and-pivot-python
df1 = nonAnaero_df.applymap(lambda x: x.split(', ') if isinstance (x, str) else [x])
#split apart the comma-separated lists of protein ids in each cell
df2 = pd.DataFrame([j for i in df1.values for j in product(*i)], columns=nonAnaero_df.columns)
#give each protein ID its own cell, while still associated with the species ID and OG

#flip the column order, putting the proteins in the first column
final_df = df2.iloc[:, ::-1]

#for speed of import into the large database, drop rows that have 'NaN' in the 'Query' column
final_df = final_df.dropna(axis=0, subset=['Query'])


#now write out the file
with open(nonAnaero_output, "w"):
    #open the output_db file for writing
    final_df.to_csv(nonAnaero_output, index=False)
```

Adding that info to the larger database (script saved in add_nonAnaero_OGs.py):

```
#!/bin/python
"""

Title: add_nonAnaero_OGs.py
Date: 19.08.2021
Author: Virág Varga

Description:
    This program adds the OG assignments to all proteins in the larger Trich_Parab
        database, using the pivoted Orthogroups.tsv file as the basis for input.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas
    numpy

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the larger trich_parab database, as well
        as the contents of the pivoted Orthogroups.tsv file, into two separate
        dataframes.
    3. Creating a combined dataframe including the OG assignments of all proteins in
        the trich_parab database.
    4. Writing out the results to a csv file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a scored database file from the Trich_Parab
        project, along with the pivoted Orthogroups.tsv file.
    - All input and output files are user-defined; the user must therefore ensure
        proper naming conventions are followed.

Usage
    ./add_nonAnaero_OGs.py input_db nonAnaero_db output_db
    OR
    python add_nonAnaero_OGs.py input_db nonAnaero_db output_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python
import numpy as np #allow manipulation of dataframe arrays


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "All_species_allOG_DB_scoreGood.csv"
#input_db contains the large input species database
nonAnaero_db = sys.argv[2]
#nonAnaero_db = "NonAnaero_OGs.csv"
#og_prot_db contains the OG pivot table to be mapped onto the input_db database
#output_db will contain the original database plus the assigned OGs
output_db = sys.argv[3]
#output_db = "All_species_allOG_Parab_DB_scoreGood.csv"


#import input databases into Pandas dataframes
#read in the trich_parab database
input_df = pd.read_csv(input_db, header = 0)
input_df.set_index('Query')
#read in the pivoted Orthogroups.tsv file
nonAnaero_df = pd.read_csv(nonAnaero_db, header = 0)
#remove the unnecessary 'Species' column
nonAnaero_df.drop('Species', axis=1, inplace=True)
nonAnaero_df.set_index('Query')
#rename the 'Orthogroup' column for easier manipulation
nonAnaero_df.rename(columns={'Orthogroup':'nonAnaero_OGs'}, inplace=True)


#now, merge the dataframes based on index
output_df = input_df.merge(nonAnaero_df, how='left')
#fill in blank cells
output_df = output_df.replace(np.nan, '-', regex=True)


#now reorder the columns
cols = list(output_df.columns.values)
#Make a list of all of the columns in the df
cols.pop(cols.index('Orthogroup'))
#Remove Orthogroup from list
cols.pop(cols.index('Score'))
#Remove Score from list
output_df = output_df[cols+['Orthogroup','Score']]
#Create new dataframe with columns in the order you want


#now write out the file
with open(output_db, "w"):
    #open the output_db file for writing
    output_df.to_csv(output_db, index=False)
```

Running the script and adding the data to the larger database:

```
#working from the DatabaseCompilation/ directory
#model:
python add_nonAnaero_OGs.py input_db nonAnaero_db output_db
#adapting it:
python add_nonAnaero_OGs.py All_species_allOG_DB_scoreGood.csv NonAnaero_OGs.csv \
  All_species_allOG_Parab_DB_scoreGood.csv
python add_nonAnaero_OGs.py All_species_allOG_DB_scoreAll.csv NonAnaero_OGs.csv \
  All_species_allOG_Parab_DB_scoreAll.csv
### and I've decided we're getting rid of the decoded files for now
### since we continue to edit these tables and I don't want/need 50 decoded files
```

Secretion profiles results files (saved in secretionProfiles.py):

```
#!/bin/python
"""

Title: secretionProfiles.py
Date: 19.08.2021
Author: Virág Varga

Description:
    This program uses a larger trich_parab database and the Orthogroups.tsv
        OrthoFinder results file in order to create files showing the secretion
        profiles of OGs.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas
    numpy
    product from itertools

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the trich_parab database and Orthogroups.tsv.
    3. Creating a score-filtered version of the trich_parab database.
    4. Creating dataframes containing the secreted OGs and their associated proteins.
    5. Writing out the results to three csv files.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a trich_parab database file and the
        Orthogroups.tsv OrthoFinder results file.
    - The Orthogroups.tsv OrthoFinder results file must be in the directory from
        which this program is run.
    - Only the trich-parab database files can be assigned names by the user from
        the command line.

Inputs & Outputs:
    - Inputs:
        - input_db: This file should be a copy of the trich_parab database that has
            not been filtered based on scoring.
        - og_db = This is the Orthogroups.tsv OrthoFinder results file.
    - Outputs:
        - output_OrthoTSV_db = This is the protein query list for the secreted OGs
            still in the format of Orthogroups.tsv (ie. rows per species per OG with
            comma-separated protein query IDs for each species).
        - output_sepProts_db = This is the protein query list for the secreted OGs
            where each query has been expanded out into its own cell.
        - output_TrichParab: This is the filtered database based on scoring.
```

```python
Usage
    ./secretionProfiles.py input_db output_TrichParab
    OR
    python secretionProfiles.py input_db output_TrichParab

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python
import numpy as np #enable manipulation of arrays
from  itertools import product


#assign command line arguments; load input and output files
#input files
input_db = sys.argv[1]
#input_db = "All_species_allOG_Parab_DB_scoreAll.csv"
og_db = "Orthogroups.tsv"
#output files
output_OrthoTSV_db = "FilteredOrthoTSV2.csv"
output_sepProts_db = "FilteredOrthoTSV_sepProts2.csv"
output_TrichParab = sys.argv[2]
#output_TrichParab = "Trich_Parab_filt_SecProf2.csv"


#read in the input trich_parab database file, assigning the first row as a header row
input_df = pd.read_csv(input_db, header=0)
#read in the input Orthogroups.tsv database file, assigning the first row as a header row
og_df = pd.read_csv(og_db, sep = '\t', header=0)


#create dataframe of filtered proteins (ie. Score >= 1)
#filt_ortho_df = input_df[input_df['Score'] >= 1]
filt_ortho_df = input_df[input_df['Score'] >= 2]
#this filtration was orignally for Score >=1, but this proved too lenient
#it resulted in a lot of false positives


#extract OGs of good proteins into a list
filt_OG_list = filt_ortho_df['Orthogroup'].tolist()
#eliminate duplicates in the list
unique_OG_list = []
[unique_OG_list.append(x) for x in filt_OG_list if x not in unique_OG_list]
#remove the '-' string indicating no OG match from the list
unique_OG_list.remove("-")

'''
#A quick sanity check:
None in unique_OG_list
#Returns:
#False
'None' in unique_OG_list
#Returns:
#False
#The printout from the list comprehension had None's in it, so I wanted to be sure
'''


#grap protein IDs associated with each OG from the Orthogroups.tsv dataframe
filt_OG_df = og_df[np.isin(og_df, unique_OG_list).any(axis=1)]
#np.isin(df, [list of strings to search])
#.any() tells it to look anywhere for a match
#.any(axis=1) ensures grabbing of the entire row where match occurred


#create a melted database where all proein queries have their own cell
pivot_OG_df = filt_OG_df.melt(id_vars="Orthogroup", var_name="Species", value_name="Query")
```

```
#pull apart the parts of the database that are comma-separated
#ie. flatten the database: each protein gets its own cell
#ref: https://stackoverflow.com/questions/50789834/parse-a-dataframe-column-by-comma-and-pivot-python
df1 = pivot_OG_df.applymap(lambda x: x.split(', ') if isinstance (x, str) else [x])
#split apart the comma-separated lists of protein ids in each cell
df2 = pd.DataFrame([j for i in df1.values for j in product(*i)], columns=pivot_OG_df.columns)
#give each protein ID its own cell, while still associated with the species ID and OG

'''
#replace the species names in df2 with the species desigantions used in the trich_parab database
filt_prot_list = df2['Query'].tolist()
#convert protein IDs column into a list

for prot_id in filt_prot_list:
    #iterate through the list of protein query IDs
    if prot_id in input_df.values:
        #check that the protein query ID is in the large database (it definitley should be)
        query_index = input_df.loc[input_df['Query'] == prot_id].index[0]
        #get the index of the row the protein query ID appears in
        species_match = input_df.iloc[query_index]['Species_ID']
        #use the index to extract the species ID for that protein
        prot_index = df2.loc[df2['Query'] == prot_id].index[0]
        #get the index of the protein query ID in the secreted OG dataframe
        df2.at[prot_index, 'Species_ID'] = species_match
        #replace the automatic species designation (ie. column names)
        #with the species/strain designation system used in the trich_parab database
'''

#The original way I tried to reorganize the species designations (above) was taking way too long
#So this is a different way to do it

#extract protein query IDs and species designations from filt_ortho_df
selected_columns = input_df[['Query', 'Species_ID']]
#select the columns that will be used to create the species reference dataframe
species_df = selected_columns.copy()
#copy those columns only into a new dataframe

#remove 'Species' column from df2
df2.drop('Species', axis=1, inplace=True)

#use the protein query IDs to join df2 and the new species_df
df3 = df2.merge(species_df, on='Query', how='left')


#create 2 results files:
# 1 file with Orthogroups.tsv formatting (ie. protein hits in 1 cell, comma-separated)
# 1 file with OG IDs in column 1, & protein query IDs in column 2 (1 protein ID per cell)
with open(output_OrthoTSV_db, "w"):
    #open the output_db file for writing
    filt_OG_df.to_csv(output_OrthoTSV_db, index=False)
with open(output_sepProts_db, "w"):
    df3.to_csv(output_sepProts_db, index=False)
#also output the filtered version of the trich_parab database
with open(output_TrichParab, "w"):
    filt_ortho_df.to_csv(output_TrichParab, index=False)
```

```
#from the DatabaseCompilation/ directory
#model:
python secretionProfiles.py input_db output_TrichParab
#adapting it:
python secretionProfiles.py All_species_allOG_Parab_DB_scoreAll.csv Trich_Parab_filt_SecProf.csv
```

Subfilter secretion profile data, to only include OGs with a PFam hit (script saved in subfilterSecretion.py):

```
#!/bin/python
"""

Title: subfilterSecretion.py
```

```
Date: 24.08.2021
Author: Virág Varga

Description:
    This program uses a larger trich_parab database and the Orthogroups.tsv
        OrthoFinder results file derivative in order to create files showing
        the secretion profiles of OGs with PFam hits.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas
    numpy
    product from itertools

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the trich_parab database and Orthogroups.tsv
        (or derivative file).
    3. Creating a PFam hit-filtered version of the trich_parab database.
    4. Creating dataframes containing the secreted OGs and their associated proteins.
    5. Writing out the results to three csv files.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a trich_parab database file and the
        Orthogroups.tsv OrthoFinder results file a derivative.
    - The Orthogroups.tsv OrthoFinder results file derivative must be in the
        directory from which this program is run.
    - Only the trich-parab database files can be assigned names by the user from
        the command line.

Inputs & Outputs:
    - Inputs:
        - input_db: This file should be a copy of the trich_parab database that has
            been filtered based on scoring.
        - og_db = This is the Orthogroups.tsv OrthoFinder results file derivative.
        - ref_db: This file should be a copy of the trich_parab database that has
            not been filtered based on scoring.
    - Outputs:
        - output_OrthoTSV_db = This is the protein query list for the secreted OGs
            still in the format of Orthogroups.tsv (ie. rows per species per OG with
            comma-separated protein query IDs for each species).
        - output_sepProts_db = This is the protein query list for the secreted OGs
            where each query has been expanded out into its own cell.
        - output_TrichParab: This is the filtered database based on scoring.

Usage
    ./subfilterSecretion.py input_db output_db
    OR
    python subfilterSecretion.py input_db output_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python
import numpy as np #enable manipulation of arrays
from  itertools import product


#assign command line arguments; load input and output files
#input files
input_db = sys.argv[1]
#input_db = "Trich_Parab_filt_SecProf2.csv"
og_db = "FilteredOrthoTSV2.csv"
ref_db = "All_species_allOG_Parab_DB_scoreAll.csv"
#output files
```

```
output_OrthoTSV_db = "FilteredOrthoTSV_PFam2.csv"
output_sepProts_db = "FilteredOrthoTSV_sepProts_PFam2.csv"
output_db = sys.argv[2]
#output_db = "Trich_Parab_filt_SecProf_PFam2.csv"


#read in the input trich_parab database file, assigning the first row as a header row
input_df = pd.read_csv(input_db, header=0)
#read in the input Orthogroups.tsv database file, assigning the first row as a header row
og_df = pd.read_csv(og_db, header=0)
#read in full trich_parab database reference file
ref_df = pd.read_csv(ref_db, header = 0)


#create dataframe of filtered proteins (ie. Score >= 1)
filt_ortho_df = input_df[input_df['PFam_Hit'] != '-']


#extract OGs of good proteins into a list
filt_OG_list = filt_ortho_df['Orthogroup'].tolist()
#eliminate duplicates in the list
unique_OG_list = []
[unique_OG_list.append(x) for x in filt_OG_list if x not in unique_OG_list]
#remove the '-' string indicating no OG match from the list
unique_OG_list.remove("-")


#grap protein IDs associated with each OG from the Orthogroups.tsv dataframe
filt_OG_df = og_df[np.isin(og_df, unique_OG_list).any(axis=1)]
#np.isin(df, [list of strings to search])
#.any() tells it to look anywhere for a match
#.any(axis=1) ensures grabbing of the entire row where match occurred


#create a melted database where all proein queries have their own cell
pivot_OG_df = filt_OG_df.melt(id_vars="Orthogroup", var_name="Species", value_name="Query")


#pull apart the parts of the database that are comma-separated
#ie. flatten the database: each protein gets its own cell
#ref: https://stackoverflow.com/questions/50789834/parse-a-dataframe-column-by-comma-and-pivot-python
df1 = pivot_OG_df.applymap(lambda x: x.split(', ') if isinstance (x, str) else [x])
#split apart the comma-separated lists of protein ids in each cell
df2 = pd.DataFrame([j for i in df1.values for j in product(*i)], columns=pivot_OG_df.columns)
#give each protein ID its own cell, while still associated with the species ID and OG


#The original way I tried to reorganize the species designations (above) was taking way too long
#So this is a different way to do it

#extract protein query IDs and species designations from filt_ortho_df
selected_columns = ref_df[['Query', 'Species_ID']]
#select the columns that will be used to create the species reference dataframe
species_df = selected_columns.copy()
#copy those columns only into a new dataframe

#remove 'Species' column from df2
df2.drop('Species', axis=1, inplace=True)

#use the protein query IDs to join df2 and the new species_df
df3 = df2.merge(species_df, on='Query', how='left')


#create 2 results files:
# 1 file with Orthogroups.tsv formatting (ie. protein hits in 1 cell, comma-separated)
# 1 file with OG IDs in column 1, & protein query IDs in column 2 (1 protein ID per cell)
with open(output_OrthoTSV_db, "w"):
    #open the output_db file for writing
    filt_OG_df.to_csv(output_OrthoTSV_db, index=False)
with open(output_sepProts_db, "w"):
    df3.to_csv(output_sepProts_db, index=False)
#also output the filtered version of the trich_parab database
with open(output_db, "w"):
    filt_ortho_df.to_csv(output_db, index=False)
```

Running it:

```
#from the DatabaseCompilation/ directory
#model:
python subfilterSecretion.py input_db output_TrichParab
#adapting it:
python subfilterSecretion.py Trich_Parab_filt_SecProf.csv Trich_Parab_filt_SecProf_PFam.csv
```

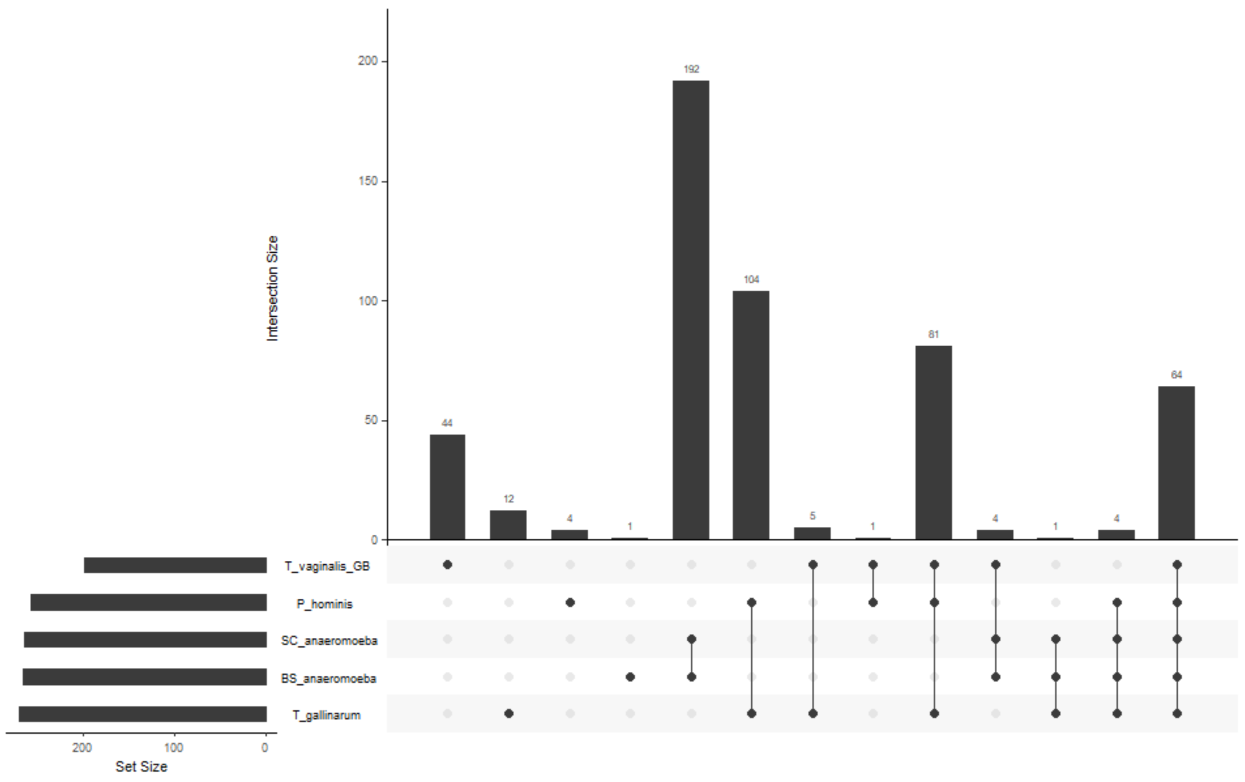
Identification of Key OGs with Count (+ R figures)

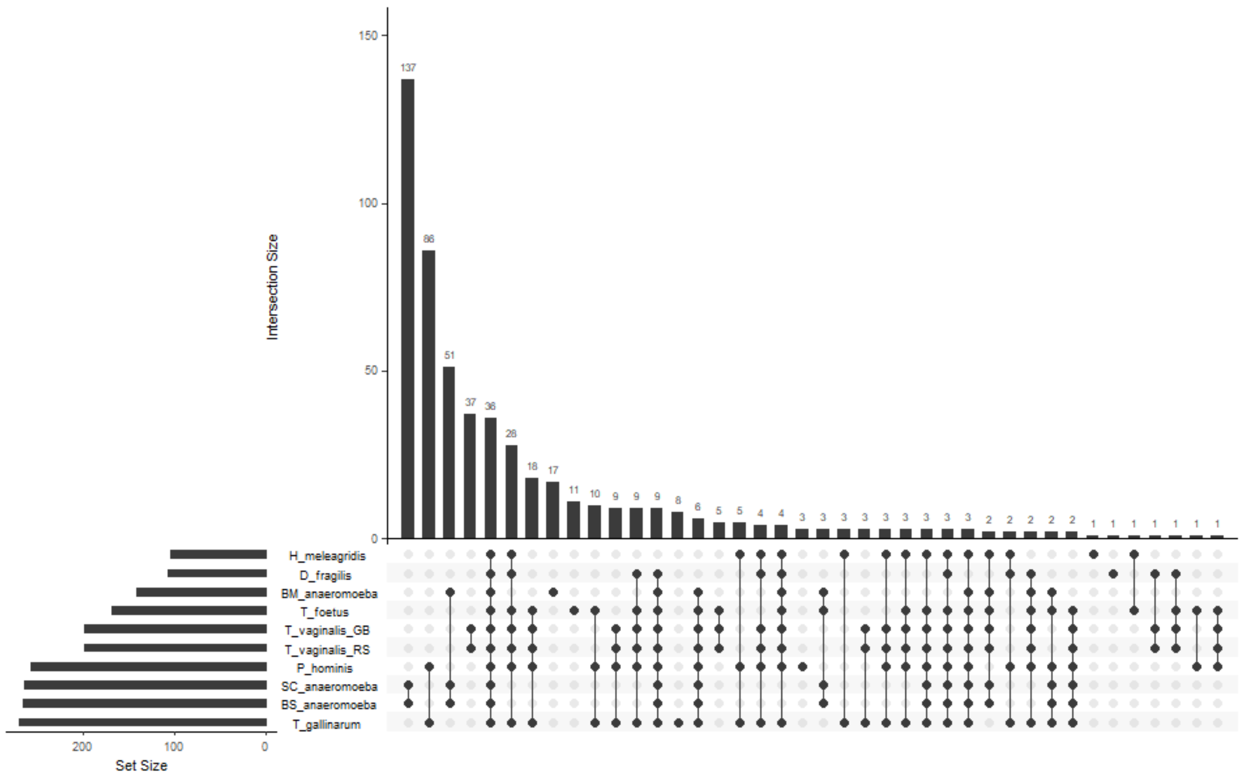
UpSetR

UpSet plots show intersections between sets of data in binary counts. The binary form of the count data was extracted from Count (above) and used as input to create UpSet plots with R package UpSetR. The UpSetR package can be installed in R easily with `install.packages("UpSetR")`. The script included below is saved to file: upSetR_etAll.R

```
# UpSetR et al.
# ie. plotting counts & frequency


#setting up the workspace
setwd("C:/Users/V/Documents/LundUni/Trich_Parab/Analyses/Figures")
#load libraries
library(UpSetR)


#load in data
#counts_df <- read.table("Filt_OG_PFam_Counts_binary__Annot.txt", header=TRUE)
counts_df <- read.table("Filt_OG_PFam_Counts2__AnnotCLEAN_Binary.txt", header=TRUE)
#can't use raw frequency data - need to use binary presence/absence data


#create UpSetR plot
#GitHub package ref: https://github.com/hms-dbmi/UpSetR
#other upset plot ref: https://jokergoo.github.io/ComplexHeatmap-reference/book/upset-plot.html
upset(counts_df)
#saved to: Count2_upsetR_START.pdf
#that worked!
#so let's try and expand on it
#count_cols <- colnames(counts_df)
#unfortunately, saving the column names to a list doesn't work for upsetR, so I had to set the names manually
upset(counts_df,
      sets = c("BM_anaeromoeba", "SC_anaeromoeba", "BS_anaeromoeba", "T_foetus", "D_fragilis", "H_meleagridis",
        "T_gallinarum", "P_hominis", "T_vaginalis_RS", "T_vaginalis_GB"),
      order.by = "freq")
#saved to: Count2_upsetR_Ver1.pdf
```

The resulting plots are quite large, and best viewed in pdf form. However, they are included below in .png format:





Count

The Trich_Parab database used as input for Count needs to be reformatted.

In order to create a file with counts of how many proteins appeared in each OG per species, to be output to a tab-delimited text file, I wrote the following program (saved in pivotCounts.py):

```python
#!/bin/python
"""

Title: pivotCounts.py
Date: 25.08.2021
Author: Virág Varga

Description:
    This program creates a pivot table of counts of proteins present in an OG per
        per species, on the basis of file that contains columns with OG IDs, protein
        IDs, and species/strain names.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the OG and protein ID database.
    3. Pivot the data table so that eeach row contains one OG ID, with information
        on the number of proteins present in that OG in each species in columns with
        the species IDs as the headers.
    4. Writing out the results to a tab-delimited text file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a database containing at least 3 columns:
        'Orthogroup', 'Query' (protein IDs in individual cells), and 'Species_ID'.

Usage
    ./pivotCounts.py input_db output_db
    OR
    python pivotCounts.py input_db output_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allow execution of code from the command line
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "FilteredOrthoTSV_sepProts_PFam2.csv"
output_db = sys.argv[2]
#output_db = "Filt_OG_PFam_Counts2.txt"

#read in the input OG database file, assigning the first row as a header row
input_df = pd.read_csv(input_db, header=0)
#input_df = input_df.replace('_', ' ', regex=True)
#replace underscores with spaces (' ')
#this is necessary as a quirk of the Count program when reading in data tables
#the above didn't work, so I put the species names in the tree file in quotation marks ("")


#create the pivot table
count_df = pd.pivot_table(data=input_df, index='Orthogroup', values= 'Query', columns='Species_ID', aggfunc='count', \
  fill_value=0)
#the Orthogroup data is used as the index, the data in the Query column is used to fille the table
#and the Species_ID categories are used as headers
#`aggfunc='count'` fills the columns with counts of unique appearances (vs. the actual protein IDs themselves)
#`fill_value=0` fills empty cells with 0 (zero) instead of NaN (don't have to do it manually later)
count_df.reset_index(inplace=True)
#reset the headers to clean up the table
#count_df.rename(columns={'Orthogroup':'Family'}, inplace=True)
#rename the Orthogroups column to match the formatting of the Count program
```

```
#the above turned out to be unnecessary - it wasn't the issue


#now write out the file
with open(output_db, "w"):
    #open the output_db file for writing
    count_df.to_csv(output_db, sep='\t', index=False)
    #the Count program needs the input data table to be in tab-delimited formatting
```

In order to add PFam annotations to the data above, I wrote the following program (saved in annotateCounts.py):

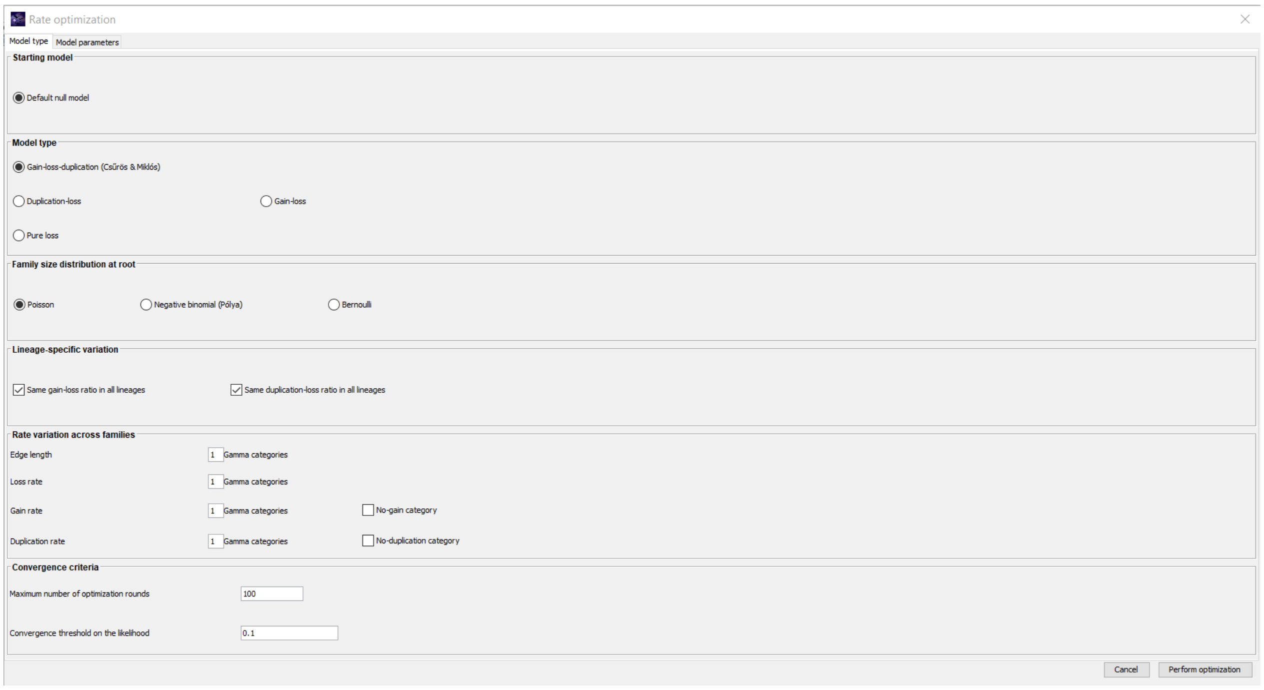
```
#!/bin/python
"""

Title: annotateCounts.py
Date: 25.08.2021
Author: Virág Varga

Description:
    This program adds annotation to the protein frequency per species per OG table
        created by program pivotCounts.py.
    At the moment, this annotation added to the database includes: PFam hits

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the database containing the counts of
        proteins per species in each OG; as well as the database containing the
        OG-PFam association data.
    3. Adding the PFam data onto the counts dataframe.
    4. Writing out the results to a tab-delimited text file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of a database containing information on the
        number of proteins per species per OG produced by program pivotCounts.py, as
        well as the OG-PFam reference database.
    - The output file name is not user-defined, but instead based on the input file's
        basename.

Usage
    ./annotateCounts.py input_db pfam_db
    OR
    python annotateCounts.py input_db pfam_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "Filt_OG_PFam_Counts2.txt"
pfam_db = sys.argv[2]
#pfam_db = "OGs2PFams_all.csv"
#pfam_db = "OGs2PFams_all_CLEAN.csv"
#output_db = ".".join(input_db.split('.')[:-1]) + '__Annot.txt'
output_db = ".".join(input_db.split('.')[:-1]) + '__AnnotCLEAN.txt'
```

```
#load databases into Pandas dataframes
input_df = pd.read_csv(input_db, sep='\t', header=0)
pfam_df = pd.read_csv(pfam_db, header = 0)


#melt together the dataframes
annot_df = input_df.merge(pfam_df, on='Orthogroup', how='left')


#write out the resulting dataframe ro the outfile
with open(output_db, "w"):
    #open the output_db file for writing
    annot_df.to_csv(output_db, sep='\t', index=False)
    #the Count program needs the input data table to be in tab-delimited formatting
```

Process of running Count analysis:

- Open program by clicking icon (requires Java)
- "Session" tab at top → "Start new session …" → select phylogenetic tree file in Newick format (in our case, SpeciesTree_nameEdit.nwk)
  - This tree file is a slightly edited version of the tree output by OrthoFinder (SpeciesTree_rooted.txt): The species names have been changed to the match the shortened versions used in the Trich_Parab database in a text editor, and the file was subsequently saved with a .nwk extension as opposed to the original .txt.
  - A phylogenetic tree will be displayed in the window; note that branch lengths are not extracted by Count from the Newick file.
- "Data" tab at top → "Open annotated table…" → selected annotated count data file (in our case, Filt_OG_PFam_Counts2__AnnotCLEAN.txt)
- "Data" tab at top → "Transform numerical profiles into binary (presence/absence) profiles"
  - By right-clicking on the new "01:Filt OG PFam Counts2 AnnotCLEAN.txt" we can save this file: "Save as…" → Filt_OG_PFam_Counts2__AnnotCLEAN_Binary.txt
- "Rates" tab at the top → "Optimize rates…" → A window opens with a number of options - the ones selected for this analysis are the defaults, pictured in the two images below.
  - After the rates have finished being optimized, a table is displayed, and the species tree showing rates of OG (gene family, in Count's terms) loss, duplication and gain will be shown. (Unfortunately, it does not appear possible to export this graphic from the program. A screenshot of the tree is included below, along with a screenshot of the rates window post-analysis.)
- "Analysis" tab at the top → "Family history by Dollo parsimony"
  - In this tab, it's possible to examine where each of these OGs was gained, lost and duplicated in the lineage.
  - Highlighting all OGs (CTRL+A) generates a summary of OG losses and gains in the entire tree.
  - From this we can say that out of a total of 548 OGs (this is the number remaining after filtering for proteins with a Score = 2; OGs associated with those proteins with PFam hits), 182 are present in the parabasalids, and 99 OGs are gained in the parabasalid lineage, specifically.
  - The results of the Dollo parsimony can bes saved in a tab-separated .txt file (I saved to Filt_OG_PFam_Counts2__AnnotCLEAN_Dollo.txt)
- ///saving and reopening below///
- To save the session: "Session" tab at the top → "Save everything …" → Assign the file a name
  - Note that the save file does not automatically provide a file extension, but this isn't an issue.
  - The file saved is in .xml format, and whether or not the user manually adds the .xml extension, Count will be able to open the session file.
  - In other words: A file named "Filt_OG_PFam_Counts2__AnnotCLEAN__Session" and a file named "Filt_OG_PFam_Counts2__AnnotCLEAN__Session.xml" can both be opened by Count, should the user forget to add the .xml extension to the file base name.
- To reopen a saved session: "Session" tab at the top → "Open previously saved session(s) …" → select the session file to be opened by Count

Rate optimization selected options:



Rate optimization

Model type | Model parameters

Starting model

[✓ Default null model]

Model type

[✓ Gain-loss-duplication (Csűrös & Miklós)]

Duplication-loss   Gain-loss

Pure loss

Family size distribution at root

[✓ Poisson]   Negative binomial (Pólya)   Bernoulli

Lineage-specific variation

[✓ Same gain-loss ratio in all lineages]   [✓ Same duplication-loss ratio in all lineages]

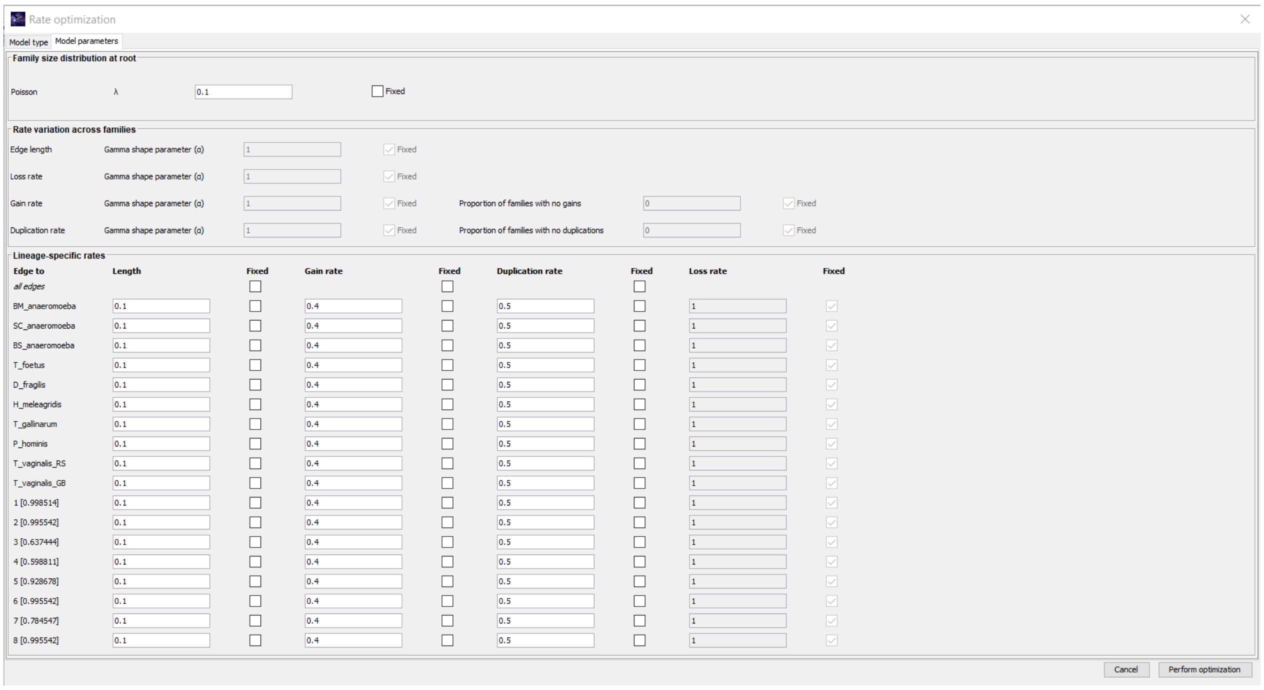
Rate variation across families

Edge length	1 Gamma categories	
Loss rate	1 Gamma categories	
Gain rate	1 Gamma categories	No-gain category
Duplication rate	1 Gamma categories	No-duplication category

Convergence criteria

Maximum number of optimization rounds	100
Convergence threshold on the likelihood	0.1

Cancel   Perform optimization



Rate optimization

Model type | Model parameters

Family size distribution at root

Poisson	λ	0.1	Fixed

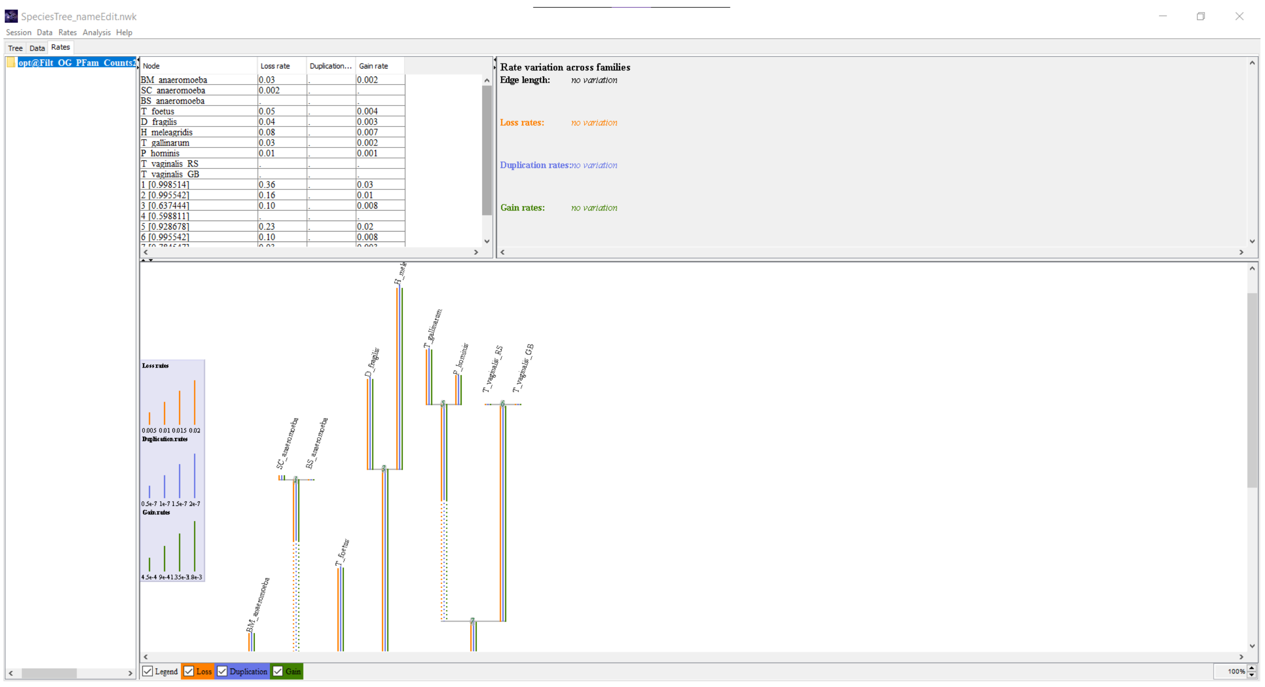
Rate variation across families

Edge length	Gamma shape parameter (α)	1	✓ Fixed			
Loss rate	Gamma shape parameter (α)	1	✓ Fixed			
Gain rate	Gamma shape parameter (α)	1	✓ Fixed	Proportion of families with no gains	0	✓ Fixed
Duplication rate	Gamma shape parameter (α)	1	✓ Fixed	Proportion of families with no duplications	0	✓ Fixed

Lineage-specific rates

Edge to	Length	Fixed	Gain rate	Fixed	Duplication rate	Fixed	Loss rate	Fixed
all edges		☐		☐		☐		
BM_anaeromoeba	0.1	☐	0.4	☐	0.5	☐	1	✓
SC_anaeromoeba	0.1	☐	0.4	☐	0.5	☐	1	✓
BS_anaeromoeba	0.1	☐	0.4	☐	0.5	☐	1	✓
T_foetus	0.1	☐	0.4	☐	0.5	☐	1	✓
D_fragilis	0.1	☐	0.4	☐	0.5	☐	1	✓
H_meleagridis	0.1	☐	0.4	☐	0.5	☐	1	✓
T_gallinarum	0.1	☐	0.4	☐	0.5	☐	1	✓
P_hominis	0.1	☐	0.4	☐	0.5	☐	1	✓
T_vaginalis_RS	0.1	☐	0.4	☐	0.5	☐	1	✓
T_vaginalis_GB	0.1	☐	0.4	☐	0.5	☐	1	✓
1 [0.998514]	0.1	☐	0.4	☐	0.5	☐	1	✓
2 [0.995542]	0.1	☐	0.4	☐	0.5	☐	1	✓
3 [0.637444]	0.1	☐	0.4	☐	0.5	☐	1	✓
4 [0.598811]	0.1	☐	0.4	☐	0.5	☐	1	✓
5 [0.928678]	0.1	☐	0.4	☐	0.5	☐	1	✓
6 [0.995542]	0.1	☐	0.4	☐	0.5	☐	1	✓
7 [0.784547]	0.1	☐	0.4	☐	0.5	☐	1	✓
8 [0.995542]	0.1	☐	0.4	☐	0.5	☐	1	✓

Cancel   Perform optimization



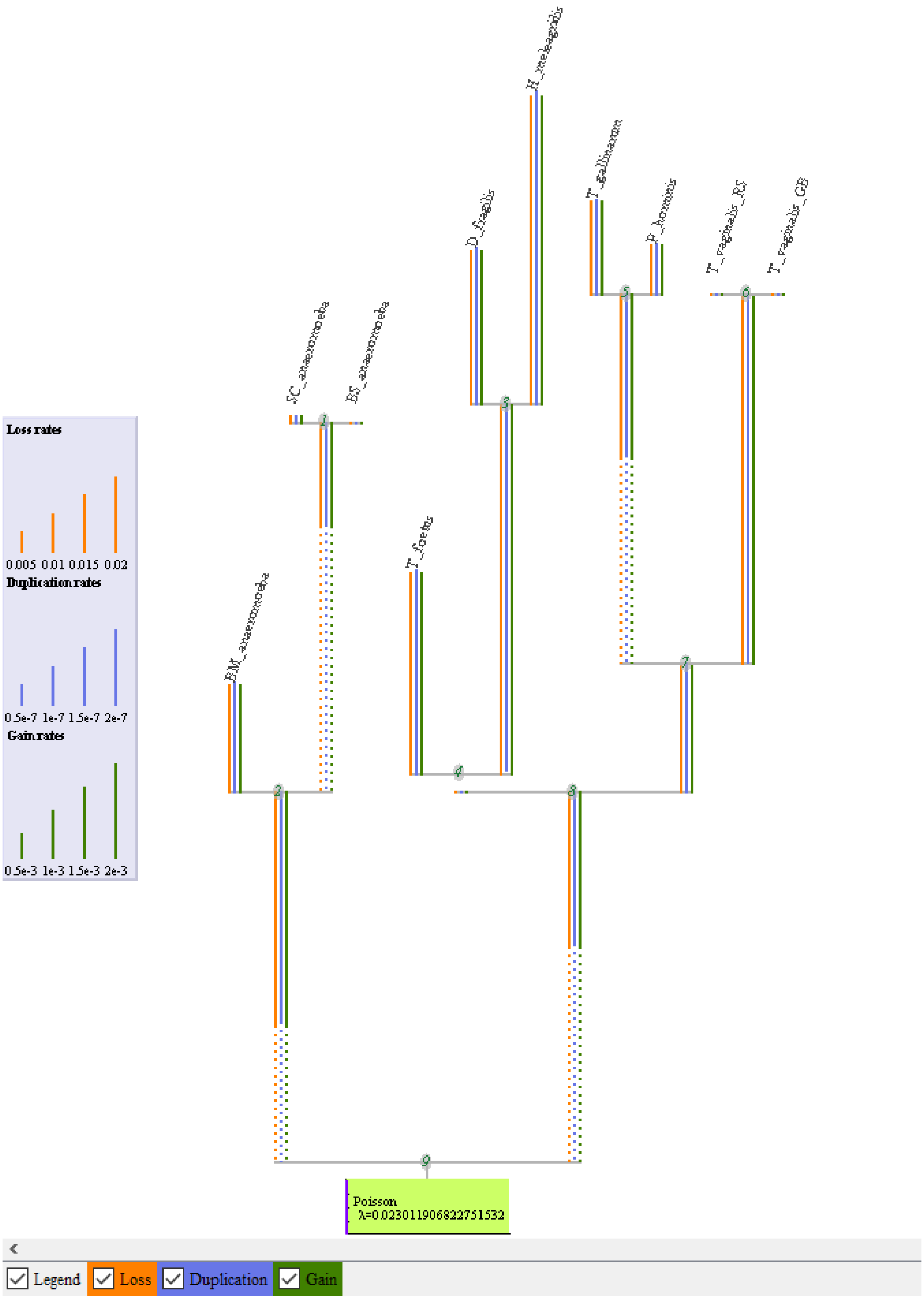
Node	Loss rate	Duplication...	Gain rate
BM_anaeromoeba	0.03	.	0.002
SC_anaeromoeba	0.002	.	.
BS_anaeromoeba	.	.	.
T_foetus	0.05	.	0.004
D_fragilis	0.04	.	0.003
H_meleagridis	0.08	.	0.007
T_gallinarum	0.03	.	0.002
P_hominis	0.01	.	0.001
T_vaginalis_RS	.	.	.
T_vaginalis_GB	.	.	.
1 [0.998514]	0.36	.	0.03
2 [0.995542]	0.16	.	0.01
3 [0.637444]	0.10	.	0.008
4 [0.598811]	.	.	.
5 [0.928678]	0.23	.	0.02
6 [0.995542]	0.10	.	0.008

Rate variation across families
Edge length: *no variation*

Loss rates: *no variation*

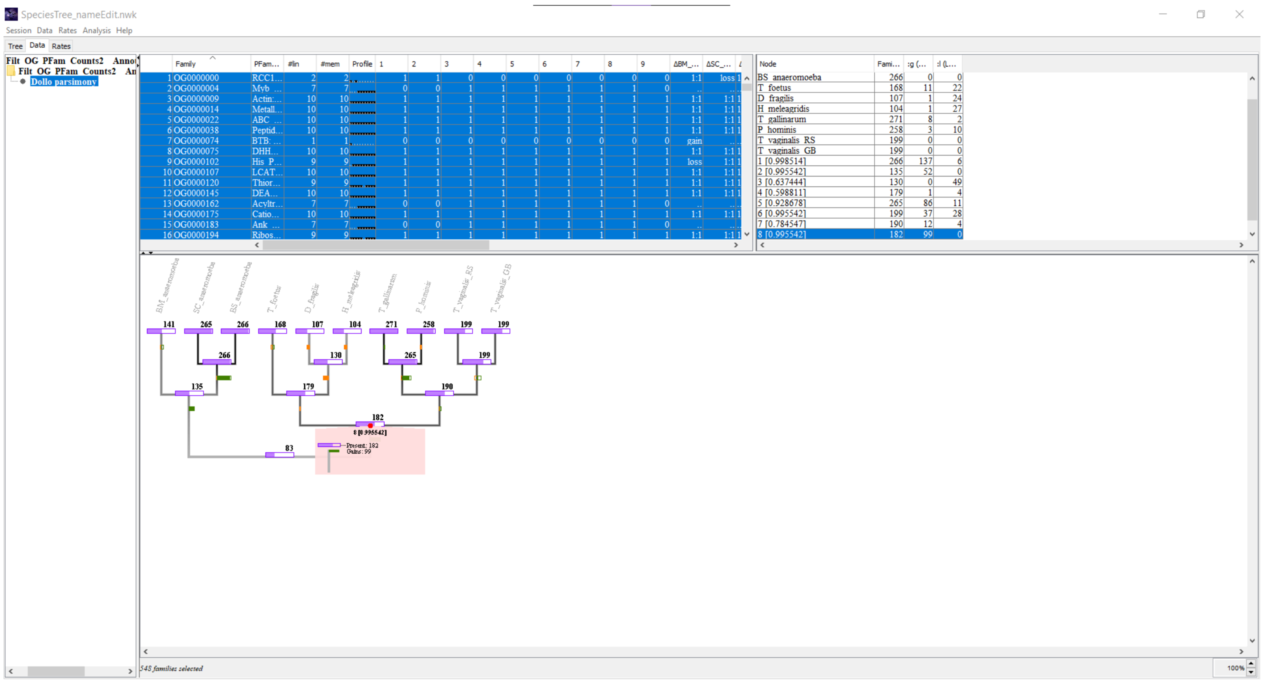
Duplication rates: *no variation*

Gain rates: *no variation*



Poisson
λ=0.023011906822751532

☑ Legend ☑ Loss ☑ Duplication ☑ Gain



	Family	PFam...	#lin	#mem	Profile	1	2	3	4	5	6	7	8	9	ΔBM_...	ΔSC_...	Δ
1	OG0000000	RCC1...	2	2		1	1	0	0	0	0	0	0	0	1:1	loss	1
2	OG0000004	Myb ...	7	7		0	0	1	1	1	1	1	1	0	..	..	..
3	OG0000009	Actin:...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
4	OG0000014	Metall...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
5	OG0000022	ABC ...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
6	OG0000038	Peptid...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
7	OG0000074	BTB: ...	1	1		0	0	0	0	0	0	0	0	0	gain	..	..
8	OG0000075	DHH...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
9	OG0000102	His P...	9	9		1	1	1	1	1	1	1	1	1	loss	1:1	1
10	OG0000107	LCAT...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
11	OG0000120	Thior...	9	9		1	1	1	1	1	1	1	1	1	1:1	1:1	1
12	OG0000145	DEA...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
13	OG0000162	Acyltr...	7	7		0	0	1	1	1	1	1	1	0	..	..	..
14	OG0000175	Catio...	10	10		1	1	1	1	1	1	1	1	1	1:1	1:1	1
15	OG0000183	Ank ...	7	7		0	0	1	1	1	1	1	1	0	..	..	..
16	OG0000194	Ribos...	9	9		1	1	1	1	1	1	1	1	1	1:1	1:1	1

Node	Fami...	:g (...	:l (L...
BS anaeromoeba	266	0	0
T foetus	168	11	22
D fragilis	107	1	24
H meleagridis	104	1	27
T gallinarum	271	8	2
P hominis	258	3	10
T vaginalis RS	199	0	0
T vaginalis GB	199	0	0
1 [0.998514]	266	137	6
2 [0.995542]	135	52	0
3 [0.637444]	130	0	49
4 [0.598811]	179	1	4
5 [0.928678]	265	86	11
6 [0.995542]	199	37	28
7 [0.784547]	190	12	4
8 [0.995542]	182	99	0

Parsing the Dollo Parsimony Data

The Dollo Parsimony data extracted from Count can be used to identify key OGs. This information was extracted from the Dollo output file using Python, below (script saved to subfilterDollo_Parab.py):

```
#!/bin/python
"""

Title: subfilterDollo_Parab.py
Date: 30.08.2021
Author: Virág Varga

Description:
    This program uses data from the Count program to filter the input file used for
        Count down to the OGs which are unique to Parabasalids.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the contents of the Count input data file and Count Dollo
        parsimony data output file.
    3. Filtering out the OGs unique to Parabasalids.
    4. After some reformatting, writing out the results to a tab-delimited text file
        which can be used as input into Dollo.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - This program requires the input of an OG database prepped as input into Count,
        along with an output file from Count containing Dollo Parsimony data.
    - While the user can determine the file names used in this code, the program is
        constructed with the intention of being used on the data collected for the
        Trich_Parab project. As a result, the node numbers used for filtration are
        specific to that data, and cannot be determined by the user from the command
        line.

Usage
    ./subfilterDollo_Parab.py input_db ref_db output_db
    OR
    python subfilterDollo_Parab.py input_db ref_db output_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments; load input and output files
#input files
input_db = sys.argv[1]
#input_db = "Filt_OG_PFam_Counts2__AnnotCLEAN_Dollo.txt"
ref_db = sys.argv[2]
#ref_db = "Filt_OG_PFam_Counts2__AnnotCLEAN.txt"
#output files
output_db = sys.argv[3]
#output_db = "Filt_OG_PFam_Counts2__AnnotCLEAN_Dollo_ParabOGs.csv"


#read in the input dollo parsimony data file, assigning the first row as a header row
input_df = pd.read_csv(input_db, skiprows=0, sep = '\t', header=1)
#the file is a tab-delimited text file, so the separator needs to be specified
#`skiprows=1` allows the first line of the file (which does not contain data or headers) to be skipped
#the header line has to be moved down 1 line as well, as a result
ref_df = pd.read_csv(ref_db, sep = '\t', header = 0)


#refromatting for ease of later use
input_df.rename(columns={'# Family':'Orthogroup'}, inplace=True)


#extract rows from the input database where the OG is present at the origin of the Parabasalids
#this is Node 8
dollo_df_1 = input_df[input_df['8'] == 1]
#dollo_df_1 contains all OGs present at Node 8 (182 total)
dollo_df_2 = dollo_df_1[dollo_df_1['9'] == 0]
#Node 9 is the root node (split between anaeramoeba and parabasalids)
#by selecting OGs not present at Node 9, we filter down to the OGs unique to parabasalids (99 total)


#extract desired OGs with PFam data from teh reference database
filt_OG_list = dollo_df_2['Orthogroup'].tolist()
#convert OG IDs column into a list
filt_ref_df = ref_df[ref_df['Orthogroup'].isin(filt_OG_list)]
#filter the reference database to only those OGs unique to parabasalids


#reformatting for ease of use
col = filt_ref_df['PFam_Dense']
#saving the PFam_Dense column to a new variable
filt_ref_df.pop('PFam_Dense')
#removing the PFam_Dense column from the original dataframe
filt_ref_df.insert(filt_ref_df.columns.get_loc('Orthogroup') + 1, col.name, col, allow_duplicates=False)
#putting the PFam_Dense column back into the dataframe, but at a different location


#print results to csv
with open(output_db, "w"):
    #open the output_db file for writing
    filt_ref_df.to_csv(output_db, index=False)
```

Create PFam function database

From the file containing the Parabasalids-only OGs, extract the PFams into an Excel table. This can then be filtered to contain only unique PFam hits using the following code (saved to unique_PFam_List.py):

```
#!/bin/python
"""

Title: unique_PFam_List.py
Date: 30.08.2021
Author: Virág Varga

Description:
    This program filters an Excel file containing a list of PFams down to only the
        unique entries in the PFam list, and outputs those into a new Excel table.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Filtering out the repeat PFam entries in the file.
    5. Writing out the filtered list to a new Excel file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.

Usage
    ./unique_PFam_List.py input_db output_db
    OR
    python unique_PFam_List.py input_db output_db

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments; load input and output files
#input files
input_db = sys.argv[1]
#input_db = "Unfilt_PFam_List.xlsx"
output_db = sys.argv[2]
#output_db = "Filt_PFam_List.xlsx"


#read in the input file, assigning the first row as a header row
input_df = pd.read_excel(input_db, header=0)


#drop duplicates
input_df.drop_duplicates('PFam_List', inplace=True)


#print results to csv
with open(output_db, "w"):
    #open the output_db file for writing
    input_df.to_excel(output_db, index=False)
```

At this stage, data on these PFams was manually extracted from the online PFam database in order to identify PFams (and associated OGs) that could prove interesting for further analysis. This determination was made based on known functions of the PFam domains, with attention to whether they are known to be associated with pathogenicity in other organisms.

PFams considered to be of interest were then traced back to the associated OGs, and input files for Count were generated based on filtration for those PFam hits. The code for the data extraction is found below (saved to subfilter_PFams.py):

```python
#!/bin/python
"""

Title: subfilter_PFams.py
Date: 25.08.2021
Author: Virág Varga

Description:
    This program filters an input file created for Count on the basis of the presence
        or absence of a given PFam. OGs with that PFam are copied into a new
        output file.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the Count input database file.
    3. Filter the input dataframe and extract OGs associated with the query PFam.
    4. Write out the results to a tab-delimited text file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - The methodology of the searching can lead to false positives, so the resulting
        file must be checked manually. Ex. searching "Peptidase_C1" may also return OGs
        containing "Peptidase_C13".
    - The output file name is not user-defined, but instead based on the input file's
        basename.

Usage
    ./subfilter_PFams.py input_db input_PFam
    OR
    python subfilter_PFams.py input_db input_PFam

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "Filt_OG_PFam_Counts2__AnnotCLEAN.txt"
input_PFam = sys.argv[2]
#input_PFam = "Gly_rich"
#input_PFam = "LRR_5"
#input_PFam = "LysM"
#input_PFam = "SH3_3"
#input_PFam = "Peptidase_M8"
#input_PFam = "Peptidase_C1"
#input_PFam = "NLPC_P60"
#input_PFam = "Mac-1"
output_db = ".".join(input_db.split('.')[:-1]) + '_' + input_PFam + '.txt'


#read in the input dollo parsimony data file, assigning the first row as a header row
#input_df = pd.read_csv(input_db, skiprows=0, sep = '\t', header=1)
#refromatting for ease of later use
#input_df.rename(columns={'# Family':'Orthogroup'}, inplace=True)
#read in the reference file with PFams
ref_df = pd.read_csv(input_db, sep = '\t', header=0)


#find OGs that contain the PFam of interest
```

```
query_PFam_df = ref_df[ref_df['PFam_Dense'].str.contains(input_PFam)]


#write out the resulting dataframe to the outfile
with open(output_db, "w"):
    #open the output_db file for writing
    query_PFam_df.to_csv(output_db, sep='\t', index=False)
    #the Count program needs the input data table to be in tab-delimited formatting
```

Gene Ontology

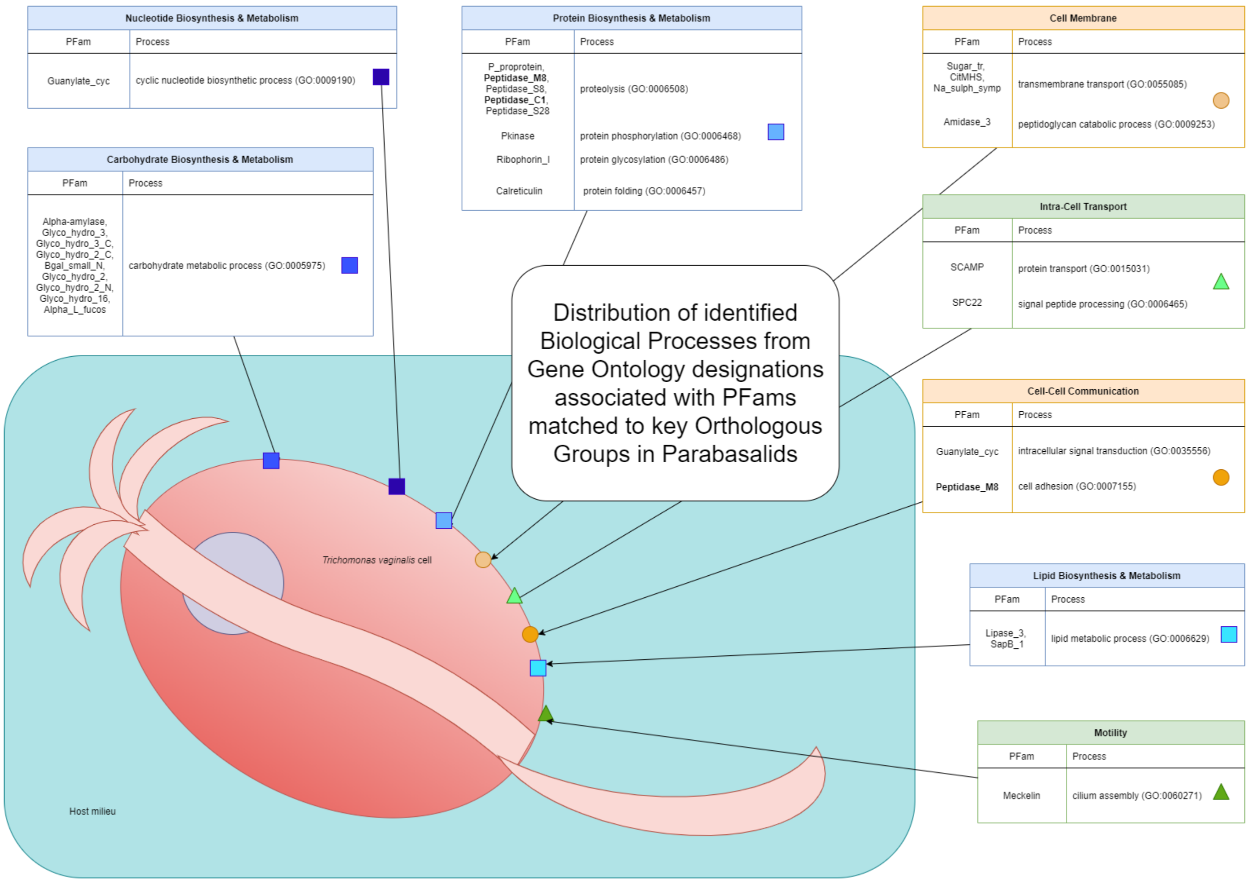
On the basis of the key PFams extracted above, further analysis was conducted on the basis of the sorts of Gene Ontology (GO) terms associated with those PFams. Below is the code to perform this filtration, saved to the file table_filterGO.py:

```
#!/bin/python
"""

Title: table_filterGO.py
Date: 09.10.2021
Author: Virág Varga

Description:
    This program is intended to filter an Excel file and extract into a new Excel file
        rows which contain a specific keyword. It was written with the intention of
        being utilized in the Trich_Parab research project to extract PFams with
        specific types of Gene Ontology designations.

List of functions:
    No functions are defined in this script.

List of standard and non-standard modules used:
    sys
    pandas

Procedure:
    1. Loading required modules; defining inputs and outputs as command line
        arguments.
    2. Using Pandas import the input database file.
    3. Filter the input dataframe and extract rows associated with the query GO term.
    4. Write out the results to a new Excel file.

Known bugs and limitations:
    - There is no quality-checking integrated into the code.
    - If the user wishes to search for a phrase, they can enter the words separated by
        underscores (_) when executing the code from teh command line. The program will
        read these underscores as spaces.
    - The output file name is not user-defined, but instead based on the input file's
        basename.

Usage
    ./table_filterGO.py input_db input_GO
    OR
    python table_filterGO.py input_db input_GO

This script was written for Python 3.8.10, in Spyder 5.0.5.

"""


#import necessary modules
import sys #allows assignment of command line arguments
import pandas as pd #facilitates manipulation of dataframes in Python


#assign command line arguments; load input and output files
input_db = sys.argv[1]
#input_db = "Filt_PFam_List__AnnotTable.xlsx"
input_GO = sys.argv[2]
#input_GO = "Biological_process"
search_GO = input_GO.replace("_", " ")
```

```
output_db = ".".join(input_db.split('.')[:-1]) + '_' + input_GO + '.xlsx'


#read in the input biological interpretation database file
ref_df = pd.read_excel(input_db, header=0)


#find OGs that contain the PFam of interest
query_GO_df = ref_df[ref_df['Gene_Ontology'].str.contains(search_GO, na=False)]
#the `na=False` flag needs to be included in the `contains()` method call
#because `contains()` cannot search through cells with NA or NaN
#it has to be told explicitly to exclude such cells from the search


#write out the resulting dataframe to the outfile
with open(output_db, "w"):
    #open the output_db file for writing
    query_GO_df.to_excel(output_db, index=False)
    #write out the results to an Excel file for ease of manual examination
```

Report Figure Creation

The data from the GO analysis conducted in the final stages of the project was visualized using the draw.io freeware. This same software was used to generate the visualized project workflow. Both figures are shown below:


Distribution of identified Biological Processes from Gene Ontology designations associated with PFams matched to key Orthologous Groups in Parabasalids
Trichomonas vaginalis cell
Host milieu

Nucleotide Biosynthesis & Metabolism

PFam	Process
Guanylate_cyc	cyclic nucleotide biosynthetic process (GO:0009190) 🟦

Protein Biosynthesis & Metabolism

PFam	Process
P_proprotein, **Peptidase_M8**, Peptidase_S8, **Peptidase_C1**, Peptidase_S28	proteolysis (GO:0006508)
Pkinase	protein phosphorylation (GO:0006468) 🟦
Ribophorin_I	protein glycosylation (GO:0006486)
Calreticulin	protein folding (GO:0006457)

Cell Membrane

PFam	Process
Sugar_tr, CitMHS, Na_sulph_symp	transmembrane transport (GO:0055085) ●
Amidase_3	peptidoglycan catabolic process (GO:0009253)

Carbohydrate Biosynthesis & Metabolism

PFam	Process
Alpha-amylase, Glyco_hydro_3, Glyco_hydro_3_C, Glyco_hydro_2_C, Bgal_small_N, Glyco_hydro_2, Glyco_hydro_2_N, Glyco_hydro_16, Alpha_L_fucos	carbohydrate metabolic process (GO:0005975) 🟦

Intra-Cell Transport

PFam	Process
SCAMP	protein transport (GO:0015031) ▲
SPC22	signal peptide processing (GO:0006465)

Cell-Cell Communication

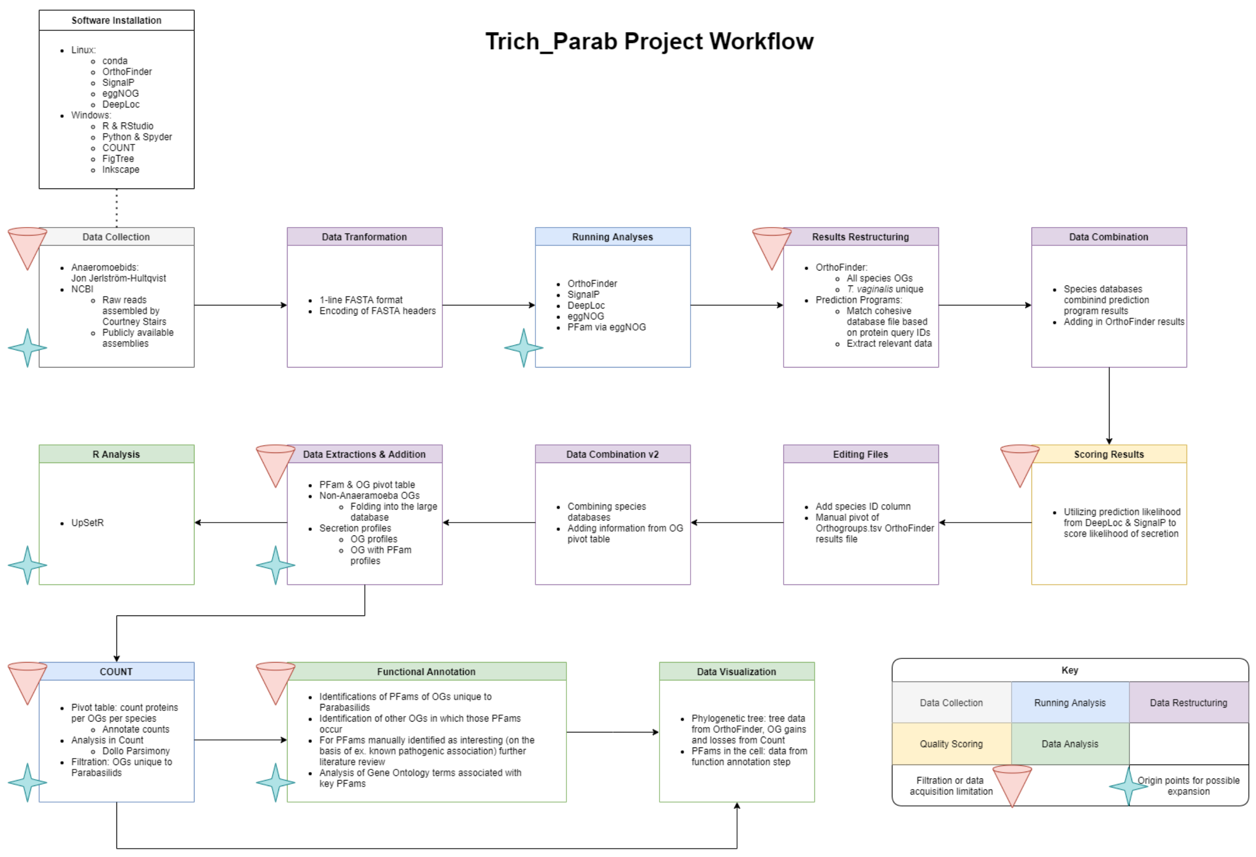
PFam	Process
Guanylate_cyc	intracellular signal transduction (GO:0035556)
Peptidase_M8	cell adhesion (GO:0007155) ●

Lipid Biosynthesis & Metabolism

PFam	Process
Lipase_3, SapB_1	lipid metabolic process (GO:0006629) 🟦

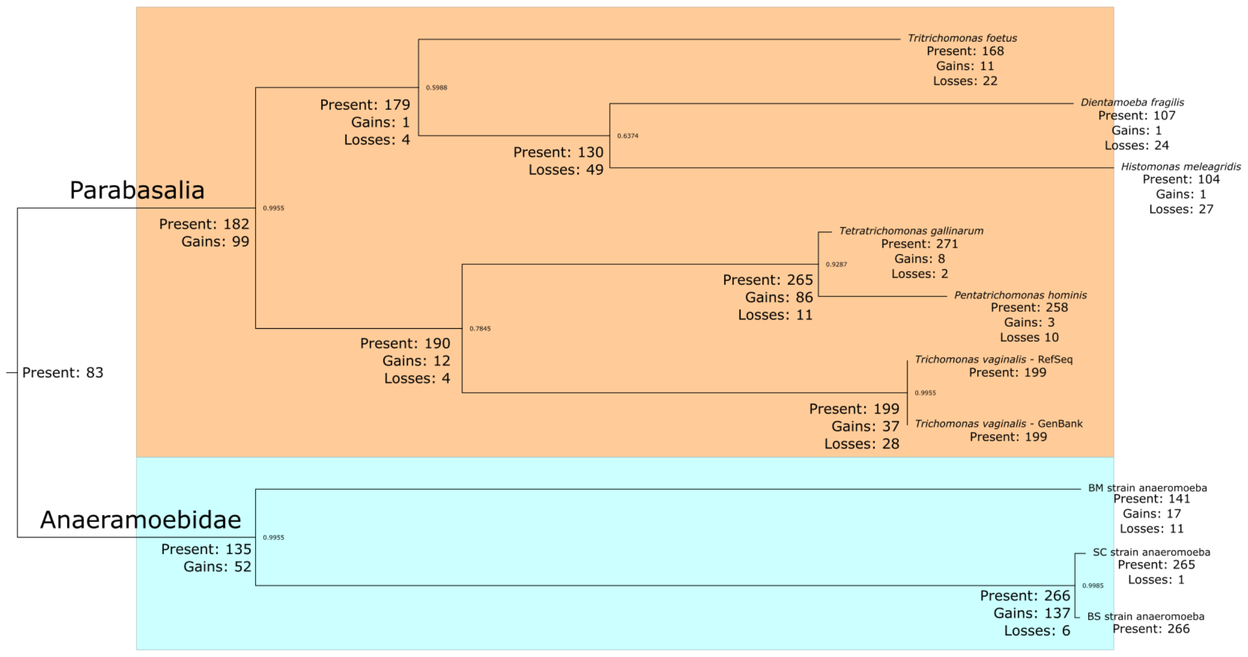
Motility

PFam	Process
Meckelin	cilium assembly (GO:0060271) ▲


Trich_Parab Project Workflow
Software Installation
Linux: conda, OrthoFinder, SignalP, eggNOG, DeepLoc
Windows: R & RStudio, Python & Spyder, COUNT, FigTree, Inkscape
Data Collection
Anaeromoebids: Jon Jerlström-Hultqvist
NCBI: Raw reads assembled by Courtney Stairs; Publicly available assemblies
Data Tranformation
1-line FASTA format
Encoding of FASTA headers
Running Analyses
OrthoFinder, SignalP, DeepLoc, eggNOG, PFam via eggNOG
Results Restructuring
OrthoFinder: All species OGs; T. vaginalis unique
Prediction Programs: Match cohesive database file based on protein query IDs; Extract relevant data
Data Combination
Species databases combinind prediction program results
Adding in OrthoFinder results
Scoring Results
Utilizing prediction likelihood from DeepLoc & SignalP to score likelihood of secretion
Editing Files
Add species ID column
Manual pivot of Orthogroups.tsv OrthoFinder results file
Data Combination v2
Combining species databases
Adding information from OG pivot table
Data Extractions & Addition
PFam & OG pivot table
Non-Anaeramoeba OGs: Folding into the large database
Secretion profiles: OG profiles; OG with PFam profiles
R Analysis
UpSetR
COUNT
Pivot table: count proteins per OGs per species; Annotate counts
Analysis in Count: Dollo Parsimony
Filtration: OGs unique to Parabasilids
Functional Annotation
Identifications of PFams of OGs unique to Parabasilids
Identification of other OGs in which those PFams occur
For PFams manually identified as interesting (on the basis of ex. known pathogenic association) further literature review
Analysis of Gene Ontology terms associated with key PFams
Data Visualization
Phylogenetic tree: tree data from OrthoFinder, OG gains and losses from Count
PFams in the cell: data from function annotation step
Key
Data Collection
Running Analysis
Data Restructuring
Quality Scoring
Data Analysis
Filtration or data acquisition limitation
Origin points for possible expansion

During the preparation of the report, a phylogenetic tree was constructed for the Supplementary Materials. The species tree created by OrthoFinder was used as the input in the phylogenetic tree visualization software FigTree, where the clades were highlighted. The tree was then exported in SVG format to be edited in Inkscape. At this stage, labels were added to the clades, and OG presence, gain and loss were added as node labels based on the data from the Dollo Parsimony analysis in Count.

The final product figure is shown below:


Parabasalia
Present: 182
Gains: 99
Anaeramoebidae
Present: 135
Gains: 52
Present: 83
Tritrichomonas foetus Present: 168 Gains: 11 Losses: 22
Present: 179 Gains: 1 Losses: 4
0.5988
Dientamoeba fragilis Present: 107 Gains: 1 Losses: 24
Histomonas meleagridis Present: 104 Gains: 1 Losses: 27
Present: 130 Losses: 49
0.6374
0.9955
Tetratrichomonas gallinarum Present: 271 Gains: 8 Losses: 2
0.9287
Pentatrichomonas hominis Present: 258 Gains: 3 Losses 10
Present: 265 Gains: 86 Losses: 11
Present: 190 Gains: 12 Losses: 4
0.7845
Trichomonas vaginalis - RefSeq Present: 199
0.9955
Trichomonas vaginalis - GenBank Present: 199
Present: 199 Gains: 37 Losses: 28
BM strain anaeromoeba Present: 141 Gains: 17 Losses: 11
0.9955
SC strain anaeromoeba Present: 265 Losses: 1
0.9985
BS strain anaeromoeba Present: 266
Present: 266 Gains: 137 Losses: 6

Software Versions

Program Versions:

- Count: version number unclear, likely 1.0
- DeepLoc: 1.0
- EggNOG: 5.0.1
- OrthoFinder: 2.5.4
- SignalP: 5.0b Linux
- Spyder: 5.0.5
    - Pandas: 1.2.3
    - Numpy: 1.20.3
- RStudio: 1.4.1717
    - UpSetR: 1.4.0

Language Versions:

- Python:
    - DeepLoc_usage `conda` environment: 3.5.1
    - trich_parab `conda` environment: 3.8.10
    - local Windows computer: 3.8.10
- R: 4.1.0

References

Mistry J, Chuguransky S, Williams L, Qureshi M, Salazar GA, Sonnhammer ELL, et al. Pfam: The protein families database in 2021. Nucleic Acids Res [Internet]. 2021 Jan 8 [cited 2021 Oct 11];49(D1):D412–9. Available from: https://academic.oup.com/nar/article/49/D1/D412/5943818

National National Center for Biotechnology Information (NCBI)[Internet]. Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information. 1988. p. https://www.ncbi.nlm.nih.gov/.

Emms DM, Kelly S. OrthoFinder: phylogenetic orthology inference for comparative genomics. [cited 2021 Jul 8]; Available from: https://doi.org/10.1186/s13059-019-1832-y

Almagro Armenteros JJ, Tsirigos KD, Sønderby CK, Petersen TN, Winther O, Brunak S, et al. SignalP 5.0 improves signal peptide predictions using deep neural networks. Nat Biotechnol 2019 374 [Internet]. 2019 Feb 18 [cited 2021 Oct 8];37(4):420–3. Available from: https://www-nature-com.ludwig.lub.lu.se/articles/s41587-019-0036-z

Almagro Armenteros JJ, Sønderby CK, Sønderby SK, Nielsen H, Winther O. DeepLoc: prediction of protein subcellular localization using deep learning. Bioinformatics [Internet]. 2017 Nov 1 [cited 2021 Oct 8];33(21):3387–95. Available from: https://academic.oup.com/bioinformatics/article/33/21/3387/3931857

Huerta-Cepas J, Szklarczyk D, Heller D, Hernández-Plaza A, Forslund SK, Cook H, et al. eggNOG 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. Nucleic Acids Res [Internet]. 2019 Jan 8 [cited 2021 Oct 8];47(D1):D309–14. Available from: https://academic-oup-com.ludwig.lub.lu.se/nar/article/47/D1/D309/5173662

pandas - Python Data Analysis Library [Internet]. 2021 [cited 2021 Mar 20]. Available from: https://pandas.pydata.org/

Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy [Internet]. Vol. 585, Nature. Nature Research; 2020 [cited 2021 Mar 20]. p. 357–62. Available from: https://doi.org/10.1038/s41586-020-2649-2

Van Rossum G, Drake FL. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace; 2009.

Raybaut P. Spyder-documentation. Available online at: pythonhosted.org. 2009;

Conway JR, Lex A, Gehlenborg N. UpSetR: an R package for the visualization of intersecting sets and their properties. Bioinformatics [Internet]. 2017 Sep 15 [cited 2021 Oct 8];33(18):2938–40. Available from: https://academic.oup.com/bioinformatics/article/33/18/2938/3884387

RStudio Team. Studio: Integrated Development for R. RStudio [Internet]. PBC, Boston, MA; 2020. Available from: http://www.rstudio.com/

R Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing [Internet]. R Foundation for Statistical Computing, Vienna, Austria; 2021. Available from: https://www.r-project.org/

Csűrös M. Count: evolutionary analysis of phylogenetic profiles with parsimony and likelihood. Bioinforma Appl Note [Internet]. 2010 Jun 15 [cited 2021 Oct 8];26(15):1910–2. Available from: https://pubmed.ncbi.nlm.nih.gov/20551134/

Blum M, Chang H-Y, Chuguransky S, Grego T, Kandasaamy S, Mitchell A, et al. The InterPro protein families and domains database: 20 years on. Nucleic Acids Res [Internet]. 2021 Jan 8 [cited 2021 Oct 11];49(D1):D344–54. Available from: https://academic.oup.com/nar/article/49/D1/D344/5958491

The Gene Ontology Consortium, Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, et al. Gene Ontology: tool for the unification of biology. Nat Genet [Internet]. 2000 May [cited 2021 Oct 19];25(1):25. Available from: /pmc/articles/PMC3037419/

The Gene Ontology Consortium. The Gene Ontology resource: enriching a GOld mine. Nucleic Acids Res [Internet]. 2021 [cited 2021 Oct 19];49(Database):D325–34. Available from: http://geneontology.org/go-cam.

Rambaut A. FigTree [Internet]. [cited 2021 May 18]. Available from: http://tree.bio.ed.ac.uk/software/figtree/

Inkscape Project. Inkscape [Internet]. Inkscape; 2020. Available from: https://inkscape.org