Motivation

There are only two complete Anuran genomes, belonging to two species of clawed frogs. *Xenopus laevis*, better known as the African Clawed frog, and *Xenopus tropicalis*, the "Western" Clawed Frog, whose territory overlaps with that of the African Clawed Frog around the "armpit" of Africa.

The NCBI (which is my favorite authority for poorly-resolved branches of the Tree of Life because when people see you using it, they assume you're too ignorant to bother arguing with you about it) lists seven families of Anurans, most of which contain species that are significantly more cute than either of these creepy tongueless weirdos.

For this assignment, I have chosen *Pelobates cultripes*, and obtained 13 RNA sequences through means that I'd prefer not to go into... I queried the NCBI's non-redundant nucleotide database "nt" for these sequences with BLASTn using the following BASH script:

```
#!/bin/bash

time nice blastn \
  -db /home/db/ncbi/nt \
  -query Pelobates_cultripes.fasta \
  -out Pelobates_cultripes-v-nt.blastn \
  -outfmt '7 std sstrand qlen slen staxids salltitles' \
  -evalue 1e-5 \
  -soft_masking true &
```

This produces a tab-delimited BLASTn output file with the following general format:

```
# BLASTN 2.2.28+
# Query: Pelobates_cultripes_01
# Database: /home/db/ncbi/nt
# Fields: query id, subject id, % identity, alignment length, mismatches, gap opens, q. start, q. end, s. start, s. end, evalue, bit score, subject strand, query length, subjec
# 500 hits found
Pelobates_cultripes_01  gi|156530256|gb|EU115993.1| 80.33   2034    369 25  1   2020    5099    7115    0.0 1511    plus    2020    17154   8345    Bombina bombina mitochondric
Pelobates_cultripes_01  gi|46242513|gb|AY585338.1|  80.32   2012    380 13  1   2005    5102    7104    0.0 1507    plus    2020    17847   8346    Bombina orientalis mitochonc
Pelobates_cultripes_01  gi|61200915|gb|AY957562.1|  80.09   2014    381 16  1   2005    5104    7106    0.0 1480    plus    2020    17173   8346    Bombina orientalis mitochonc
Pelobates_cultripes_01  gi|55274401|gb|AY789013.1|  79.74   2034    376 23  1   2002    5149    7178    0.0 1441    plus    2020    17610   8364    Xenopus tropicalis mitochonc
Pelobates_cultripes_01  gi|304556737|gb|HM991335.1| 79.36   2045    376 38  1   2012    5158    7189    0.0 1397    plus    2020    17717   8355    Xenopus laevis mitochondrior
```

We're going to write a Python program that will allow us to filter these results using something more sophisticated than simply "Field X has a value greater or less than Y", which could honestly be handled more easily and quickly using something like AWK or a Perl one-liner (use the right tool for each job!).

I've decided to extend the popular minimum percent identity filter to require a minimum percent of the *total* number of nucleotides in the query sequence be aligned, rather than just a minimum percent of the nucleotides within the aligned region. To accomplish this, we first need to determine the number of individual nucleotides that were aligned between the two sequences:

$$AlignedNucleotides = AlignmentLength \times PercentIdentity$$

We will then divide this by the query length to get the fraction of the total nucleotides in the query sequences that were found by BLASTn to be identical to those in the subject sequence.

$$GlobalPercentIdentity = \frac{AlignedNucleotides}{QueryLength}$$

Python Code

File IO

Unless we want to hard code the entire table of BLASTn results into our program, we will need to be able to read and write entire files

In [ ]:
```
#!/usr/local/env python

InFile = open("Pelobates_cultripes-v-nt.blastn", "r")
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

for Line in InFile:
    OutFile.write(Line)

InFile.close()
OutFile.close()
```

Parsing input lines using a delimiter character

These input files are tab-delimited, so to separate them into their individual fields, we can use the special .split() function that is built into all string variables

In [ ]:
```
#!/usr/local/env python

InFile = open("Pelobates_cultripes-v-nt.blastn", "r")
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

for Line in InFile:
    Temp = Line.split()

InFile.close()
OutFile.close()
```

This doesn't actually print anything though, so let's try storing the fields we're going to be using in variables with meaningful (abbreviated) names, explicitely casting each one into an appropriate type, and then print them

In [ ]:
```
#!/usr/bin/env python

InFile = open("Pelobates_cultripes-v-nt.blastn", "r")
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

for Line in InFile:
    Temp = Line.split('\t')

    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    OutTuple = (QryID, SubID, PercID, AlnLen, QryLen, SubLen)
    OutLine = "%s\t%s\t%.1f\t%s\t%s\t%s\n" % OutTuple
    OutFile.write(OutLine)

InFile.close()
OutFile.close()
```

Oops! Comment lines in input files can be a real pain, so let's skip them!

In [ ]:
```
#!/usr/bin/env python

InFile = open("Pelobates_cultripes-v-nt.blastn", "r")
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

for Line in InFile:

    if Line[0] == "#":
        continue

    Temp = Line.split('\t')

    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    OutTuple = (QryID, SubID, PercID, AlnLen, QryLen, SubLen)
    OutLine = "%s\t%s\t%.1f\t%s\t%s\t%s\n" % OutTuple
    OutFile.write(OutLine)

InFile.close()
OutFile.close()
```

Printing a custom header line

I find header lines useful, and since I tend to pipe my Python output into R, I try to choose names that would work reasonably well as column names in an R data frame

In [ ]:
```
#!/usr/bin/env python

InFile = open("Pelobates_cultripes-v-nt.blastn", "r")
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

OutFile.write("QryID\tSubID\tPercID\tAlnLen\tQryLen\tSubLen\n")

for Line in InFile:

    if Line[0] == "#":
        continue

    Temp = Line.split('\t')

    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    OutTuple = (QryID, SubID, PercID, AlnLen, QryLen, SubLen)
    OutLine = "%s\t%s\t%.1f\t%s\t%s\t%s\n" % OutTuple
    OutFile.write(OutLine)

InFile.close()
OutFile.close()
```

Beautiful!

Computing custom statistics

I have also updated the header and output lines to contain the new custom statistics

In [ ]:
```
#!/usr/bin/env python

InFile = open("Pelobates_cultripes-v-nt.blastn", "r")
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

OutFile.write("QryID\tSubID\tPercID\tAlnLen\tAlnNucs\tQryLen\tGlbPercID\tSubLen\n")

for Line in InFile:

    if Line[0] == "#":
        continue

    Temp = Line.split('\t')

    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    AlnNucs = PercID * AlnLen / 100
    GlbPercID = 100 * AlnNucs / QryLen

    OutTuple = (QryID, SubID, PercID, AlnLen, AlnNucs, QryLen, GlbPercID, SubLen)
    OutLine = "%s\t%s\t%.1f\t%s\t%s\t%.1f\t%.1f\t%s\n" % OutTuple
    OutFile.write(OutLine)

InFile.close()
OutFile.close()
```

If you look carefully at the output at this point, you should notice a couple of things. First, I lied to you. The alignment length reported by BLAST is the longer of the regions from either sequence, which means that the aligned region can be longer than the query sequence. Second, when the alignment length is roughly the same as the query sequence length, rounding errors can cause the global percent identity to be slightly longer than the local percent identity reported by BLAST.

Comment your code

Do it. You'll thank yourself later.

In [ ]:
```
#!/usr/bin/env python

# Open an input file of tab-delimited BLASTn results
InFile = open("Pelobates_cultripes-v-nt.blastn", "r")

# Open a file for output
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

# Print a header line
OutFile.write("QryID\tSubID\tPercID\tAlnLen\tAlnNucs\tQryLen\tGlbPercID\tSubLen\n")
```

```
# Iterate through the file, one line at a time
for Line in InFile:

    # Skip comment lines
    if Line[0] == "#":
        continue

    # Split each tab-delimited line into a temporary list variable
    Temp = Line.split('\t')

    # Store useful fields in variables with meaningful names
    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    # Compute some custom statistics
    AlnNucs = PercID * AlnLen / 100
    GlbPercID = 100 * AlnNucs / QryLen

    # Print the bits we care about using ~*fancy formatting*~
    OutTuple = (QryID, SubID, PercID, AlnLen, AlnNucs, QryLen, GlbPercID, SubLen)
    OutLine = "%s\t%s\t%.1f\t%s\t%s\t%.1f\t%.1f\t%s\n" % OutTuple
    OutFile.write(OutLine)

# Close files when you're done using them
InFile.close()
OutFile.close()
```

Unfortunately, individually commenting every little piece of code like this can disrupt the visual layout of the overall program. Jonathan and/or Keith will show you better ways to comment Python code later in the semester, but you'll have to learn a few other concepts before then, so just do them individually for now.

(Finally!) Filtering results

Let's put these custom statistics to use

In [ ]:
```
#!/usr/bin/env python

# Open an input file of tab-delimited BLASTn results
InFile = open("Pelobates_cultripes-v-nt.blastn", "r")

# Open a file for output
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

# Print a header line
OutFile.write("QryID\tSubID\tPercID\tAlnLen\tAlnNucs\tQryLen\tGlbPercID\tSubLen\n")

# Global minimum percent identity threshold
MinGlbPercID = 70

# Iterate through the file, one line at a time
for Line in InFile:

    # Skip comment lines
    if Line[0] == "#":
        continue
```

```
    # Split each tab-delimited line into a temporary list variable
    Temp = Line.split('\t')

    # Store useful fields in variables with meaningful names
    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    # Compute some custom statistics
    AlnNucs = PercID * AlnLen / 100
    GlbPercID = 100 * AlnNucs / QryLen

    # Only print results that meet or exceed our minimum threshold
    if GlbPercID >= MinGlbPercID:

        # Print the bits we care about using ~*fancy formatting*~
        OutTuple = (QryID, SubID, PercID, AlnLen, AlnNucs, QryLen, GlbPercID, SubLen)
        OutLine = "%s\t%s\t%.1f\t%s\t%.1f\t%s\t%.1f\t%s\n" % OutTuple
        OutFile.write(OutLine)

# Close files when you're done using them
InFile.close()
OutFile.close()
```

Combining filters

Since we've gone to the effort of setting this up, we might as well set up the same filter for the subject sequences

In [ ]:

```
#!/usr/bin/env python

# Open an input file of tab-delimited BLASTn results
InFile = open("Pelobates_cultripes-v-nt.blastn", "r")

# Open a file for output
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

# Print a header line
OutFile.write("QryID\tSubID\tPercID\tAlnLen\tAlnNucs\tQryLen\tGlbQryPercID\tSubLen\tGlbSubPercID\n")

# Global minimum percent identity threshold
MinGlbPercID = 70

# Iterate through the file, one line at a time
for Line in InFile:

    # Skip comment lines
    if Line[0] == "#":
        continue

    # Split each tab-delimited line into a temporary list variable
    Temp = Line.split('\t')

    # Store useful fields in variables with meaningful names
    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
```

```
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    # Compute some custom statistics
    AlnNucs = PercID * AlnLen / 100
    GlbQryPercID = 100 * AlnNucs / QryLen
    GlbSubPercID = 100 * AlnNucs / SubLen

    # Only print results that meet or exceed our minimum threshold
    if GlbQryPercID >= MinGlbPercID and GlbSubPercID >= MinGlbPercID:

        # Print the bits we care about using ~*fancy formatting*~
        OutTuple = (QryID, SubID, PercID, AlnLen, AlnNucs, QryLen, GlbQryPercID, SubLen, GlbSubPercID)
        OutLine = "%s\t%s\t%.1f\t%s\t%.1f\t%s\t%.1f\t%s\t%.1f\n" % OutTuple
        OutFile.write(OutLine)

# Close files when you're done using them
InFile.close()
OutFile.close()
```

Hmm, there don't seem to be very many hits that have captured at least 70% of both sequences, by which I mean none. Before we go in search of a more appropriate cutoff though, let's make it easier to play with that parameter by importing the "sys" module and setting it equal to a command line argument.

Command line arguments

In [ ]:

```
#!/usr/bin/env python

import sys

# Open an input file of tab-delimited BLASTn results
InFile = open("Pelobates_cultripes-v-nt.blastn", "r")

# Open a file for output
OutFile = open("Pelobates_cultripes-v-nt.filtered.blastn", "w")

# Print a header line
OutFile.write("QryID\tSubID\tPercID\tAlnLen\tAlnNucs\tQryLen\tGlbQryPercID\tSubLen\tGlbSubPercID\n")

# Get global minimum percent identity threshold from user
MinGlbPercID = float(sys.argv[1])

# Iterate through the file, one line at a time
for Line in InFile:

    # Skip comment lines
    if Line[0] == "#":
        continue

    # Split each tab-delimited line into a temporary list variable
    Temp = Line.split('\t')

    # Store useful fields in variables with meaningful names
    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])

    # Compute some custom statistics
```

```
    AlnNucs = PercID * AlnLen / 100
    GlbQryPercID = 100 * AlnNucs / QryLen
    GlbSubPercID = 100 * AlnNucs / SubLen

    # Only print results that meet or exceed our minimum threshold
    if GlbQryPercID >= MinGlbPercID and GlbSubPercID >= MinGlbPercID:

        # Print the bits we care about using ~*fancy formatting*~
        OutTuple = (QryID, SubID, PercID, AlnLen, AlnNucs, QryLen, GlbQryPercID, SubLen, GlbSubPercID)
        OutLine = "%s\t%s\t%.1f\t%s\t%.1f\t%s\t%.1f\t%s\t%.1f\n" % OutTuple
        OutFile.write(OutLine)

# Close files when you're done using them
InFile.close()
```

Now when we run our program, we need to provide it with a cutoff as a command line argument.

In [ ]:
```
%run filter_tabbed_blast.py 60
```

60% gets us a few hits for most sequences, but none for Pelobates cultripes sequences number 2, 4, 6-9, 12, & 13

In [ ]:
```
%run filter_tabbed_blast.py 50
```

50% adds at least one hit for Pelobates cultripes sequences number 8, 12 & 13, but still nothing for 2, 4, 6, 7, & 9. This is shaping up to be an annoying biology problem, but at least the program has gotten easy to re-run. Let's also modify it so that we can easily change the input and output files at run time.

In [ ]:
```
#!/usr/bin/env python

import sys

# Open an input file of tab-delimited BLASTn results
InFile = open(sys.argv[1], "r")

# Open a file for output
OutFile = open(sys.argv[2], "w")

# Print a header line
OutFile.write("QryID\tSubID\tPercID\tAlnLen\tAlnNucs\tQryLen\tGlbQryPercID\tSubLen\tGlbSubPercID\n")

# Get global minimum percent identity threshold from user
MinGlbPercID = float(sys.argv[3])

# Iterate through the file, one line at a time
for Line in InFile:

    # Skip comment lines
    if Line[0] == "#":
        continue

    # Split each tab-delimited line into a temporary list variable
    Temp = Line.split('\t')

    # Store useful fields in variables with meaningful names
    QryID = str(Temp[0])
    SubID = str(Temp[1])
    PercID = float(Temp[2])
    AlnLen = int(Temp[3])
    QryLen = int(Temp[13])
    SubLen = int(Temp[14])
```

```
    # Compute some custom statistics
    AlnNucs = PercID * AlnLen / 100
    GlbQryPercID = 100 * AlnNucs / QryLen
    GlbSubPercID = 100 * AlnNucs / SubLen

    # Only print results that meet or exceed our minimum threshold
    if GlbQryPercID >= MinGlbPercID and GlbSubPercID >= MinGlbPercID:

        # Print the bits we care about using ~*fancy formatting*~
        OutTuple = (QryID, SubID, PercID, AlnLen, AlnNucs, QryLen, GlbQryPercID, SubLen, GlbSubPercID)
        OutLine = "%s\t%s\t%.1f\t%s\t%.1f\t%s\t%.1f\t%s\t%.1f\n" % OutTuple
        OutFile.write(OutLine)

# Close files when you're done using them
InFile.close()
OutFile.close()
```

Now we run the program like so:

In [ ]:
```
%run filter_tabbed_blast.py Pelobates_cultripes-v-nt.blastn Pelobates_cultripes-v-nt.filtered.blastn 50
```

Homework

Another statistic that can be useful for filtering is the bitscore, except that flat bitscores aren't easily comparable across sequences. One option for normalizing bitscores is to divide them by either the alignment length or the query length. In keeping with the theme of this document, let's use the full query length and define the statistic as the "bit per base".

$$BitPerBase = \frac{BitScore}{QueryLength}$$

You will write a program that computes this statistic and uses it to filter the tab-delimited BLASTn output file used in the examples above. Your program will accept the following three command line arguments, in order:

I. Input file containing tab-delimited BLASTn results
II. Output file name
III. Minimum bit per base score

Your program will print the following fields, starting with a descriptive header line:

I. Query ID
II. Subject ID
III. Bit score
IV. Query length
V. Bit per base (rounded to three decimal places)

I don't want to step on Jonathan's toes, but I realize I'm late giving out this assignment. I intend to grade these on Thursday. When I start grading, whatever files I find on Grace or in my inbox will be accepted. I will not respond to questions about exactly what time that's likely to be, nor will I send out an announcement when I begin grading. If you want to be safe, submit your Python script before you go to bed Wednesday night.