**Code Peer Review**

María Cadavid Vélez

**Instructions**:

Today, researchers usually create fragments of code as part of the analysis they made during their scientific activities. In the research paper, Franco‐Sierra, Nicolás D., and Juan F. Díaz‐Nieto. “Rapid Mitochondrial Genome Sequencing Based on Oxford Nanopore Sequencing and a Proxy for Vertebrate Species Identification.” Ecology and Evolution 10, no. 7 (April 2020): 3544–60. <https://doi.org/10.1002/ece3.6151>, Franco presents a Python code you must analize: <https://github.com/nidafra92/squirrel-project/blob/master/mtblaster.py>

**Code Review:**

In this paper they wanted to evaluate the usefulness of the sequencer MinION from Oxford Nanopore technologies for rapid species identification, even applicable to *on site* experiments with vertebrates, and this way avoid the collection of specimens that are not really needed to be collected. The code they implemented for this objective identifies sequences that presumably correspond to mtDNA with a BLAST-based strategy.

The script they wrote requires as input data the fastq file that is obtained from the sequencing and a database of mitochondrial DNA which is used to compare against to. The code consists of basically 4 steps: Initially it collects the reads from the sequencer and filters them by length using a max length of 1700bp in this case. Second it takes the filtered reads and runs a blast command to align the reads against the local database provided. Then, it parses the results from the alignment to count the total hits and the number of reads with hits. Finally, it creates a new fastq file containing only the reads that reported mitochondrial hits.

The code is pretty straight forward, well documented and easy to understand. It its written like a story, step by step. I like a lot the way Franco introduces prints in the code to inform the user what the program is doing, in case of an error, having this kind of prints will easily help the user find where the error was. Besides, it is clear what kind of inputs the program needs, because the input variables are very descriptive and short at the same time. Also, the rest of the variables are explicit as well as indicative of what kind of data they are supposed to contain.

The way Franco use available libraries is very smart, I didn’t know that the *blast command line* was part of the Bio library, this new discovery will simplify my future work a lot, allowing me to run the process of the filter and blast together in only one straight and simple code. Furthermore, how Franco uses the “*with*” statement to handle the files makes the code much more readable and cleaner, this is definitively a good practice I could implement to reduce the length and complexity of my codes instead of opening and closing files.

Although, it is well written and understandable, there is small change I would recommend, and it is the use of functions for each of the previous indicated steps. I would have done 4 functions with meaningful verbal names: one for the filtering of the reads, one for the alignment with blast, one for the parsing, and one for writing the new file.

In conclusion, this code is very well written, complete and simple to understand while it fulfills its complex objective of identifying the reads with mitochondrial hits. A very good example of clean and friendly code.