1. Hello, my name is Allen Cao and today I will be discussing how we decided to use two programming languages: Java and Python.
2. Our goals for this project include querying and extracting epitope information from the IEDB database, blasting them against their reference proteomes and filtering results, then blasting their antigens against the reference proteome, all to make sure that these epitopes are accurate. Once filtered, we can use these epitopes in studies, namely about conservancy and disorder.
3. I had already taken AP computer science in school, which teaches only Java, so in the beginning I turned to Java as my go-to language. However, while I went to Java as I was more familiar with it, Python eventually became the better language for our project. While Java is more robust, efficient, and faster, Python suits smaller projects such as ours and is easier to code with. Python’s syntax accommodates short programs more, as indenting is used for formatting. Java requires more code to get the same work done, which is not the goal.
4. To accomplish much of what we wanted done, a bioinformatics package would be very convenient. Biojava and Biopython are the two popular add-ons to each language, and provide many useful functions. However, Biojava is strangely hard to install, with multiple versions of Biojava and Eclipse making things difficult. 1.8 Biojava and 3.0 Biojava along with Kepler, Juno, and all the other versions of Eclipse have different instructions. One installation method requires Maven, and adding many plugins to Eclipse. Biopython, on the other hand, is an extremely simple click and install.
5. Querying the IEDB database can be done in Java or Python, provided the database is downloaded. However, the fasta files for the epitopes only need to be generated once, so MySQL Workbench already provides the needed functionality. Generating a csv and converting it to a fasta file is easily done through Python or Java, and each have their respective imports. However, there might be a need to re-query information from IEDb to access additional information, and Python is slightly more user-friendly with a MySQL package, not just SQL.
6. The next step was BLASTing the fastas against their respective proteomes, and later the antigens against the proteomes. Programmatic access is definitely needed for this, so we downloaded blast and installed it as the easiest method to BLAST proteins. The downloadable blast is run through command line, which is easily executable in both Python and Java. However, BioPython can import NcbiblastpCommandLine, which makes blasting slightly easier. Once again, Python proves to be slightly more convenient.
7. Finally, we parsed through blast results. The blast output could come in multiple forms. A search of how to parse blast output in Java gives confusing instructions on generating sax events and creating Java objects, but this is for raw blast output. Blast defaults to a txt file, so I originally tunnel-visioned into parsing this file. I wrote a custom text parser in Java, which I originally thought would be simpler but turned out to be a monstrosity in code. The alternative was much easier: outputting in xml or csv provides easily-accessible information. BioPython even has a blast xml parser, the uml Diagram of which is here. Parsing either format proved to be much easier than the text parser, which did work, though.
8. As a whole, Python seemed much simpler to implement what we wanted. Java was still useful for small bits that proved difficult in Python, but most programs suited the small scripts of Python more.