Exercise 2

Questions

1. Match the terms and definitions on gradescope
2. Select the best answer on gradescope.
3. (18 points 8 for script, 10 for short answer)

   This question is composed of three parts:

   ```
    1. A portion (that is not assessed) where you work in your terminal
   to make sure you understand the concepts you are working with.
    2. A portion where you submit a script.
    3. A short answer question.
   ```

   For this question you are going to be asked to create a variable and an alias and to answer a question about the functional implications of the syntax you use to do so. In order to answer this question correctly, you should work in your terminal to ensure that you are able to create and use Bash variables and aliases.

   Variables are very useful in both interactive terminal sessions and in scripts. Aliases, however are generally only used in interactive terminal sessions. That's why I encourage you to experiment in a terminal as part of this question. The submission of a script for this question is intended to allow any of you who haven't completed script-based coding assignments before to start with the submission of a very simple script. Please let me or your TAs know if you have any issues submitting your script.

   Now for the three parts of the question:

   1. For part 1 (which is ungraded and just to help you to experiment with the assignment material - you'll be expected to guide your own experimentation for all other assignments) perform the following steps:

      - Create a variable called "wizard" which has the value "Gandalf the Grey" (Note the capitalization and British English spelling of grey/gray, which the autograder will expect).
      - Make sure you know how to print the contents of the wizard variable to your terminal using a Bash command. Your command should print only the contents of the wizard variable and no other variables (i.e., don't just view all variables in your environment with `env`).
      - Create an alias called `view_wizard` for the command you used to view the contents of you wizard variable. Note that your alias should actually use the wizard variable itself. You should not simply write out "Gandalf the Grey" anywhere within in your alias definition.
      - Confirm that you are able to use your `view_wizard` alias to print the contents of the wizard variable.

- Note whether you used single (') or double quotes (") in the definition of your `view_wizard` alias. Now create another alias which uses the other kind of quotes. Try both aliases and take note of whether there is any difference in behaviour.
- Now assign a new value to your wizard variable, such as "Radagast the Brown".
- Run each of the two aliases you have created (i.e., the version using single vs double quotes). Note any difference in behaviour.

2. For part 2 you need to submit a script to demonstrate that you know the syntax for creating a variable and an alias. You should submit a script with the filename "wizard.sh" to the programming assignment which was assigned along with this quiz. The script should contain two lines of code. Line 1 should be the command to create a variable called "wizard" with the value "Gandalf the Grey". The second line should be the command to create an alias called "view_wizard" for a command to view the contents of the wizard variable.

3. For part 3 answer the following question in the short answer question assignment on Gradescope:

   The commands in your submitted script create a variable called wizard and an alias called view_wizard which prints the contents of the wizard variable. Consider a situation in which the value stored in "wizard" may change (e.g., to "Radagast the Brown"). 1. Describe function performed by quotes when creating the alias. 2. Is there a difference in the behavior of the alias if you create the alias using single quotes or double quotes? If so, explain the difference. If not, why not?

4. "dir_script.sh" (10 points)

   Write a script called "dir_script.sh" which performs the following steps. The usage of the script should be `dir_script.sh <path>`. (please ask me or a TA if you are unsure about exactly what that usage specification means). The steps:

   1. Take a path to a directory as command line input
   2. Make the specified directory, as well as any missing parent directories
   3. Enter the new directory
   4. print the absolute path of the new working directory to stdout

5. "change_headers.sh" (22 points)

   The next two questions will be linked as you will be using the same data to answer them both. Each question will begin with a description of the data you will be manipulating and a explanation of why you are performing the manipulation. These scripts are genuinely useful in a bioinformatic analysis and we shall develop them over the coming weeks using the concepts we cover in class.

   The following section is a description of FASTA format data. If you are already familiar with FASTA format sequence data, all the information required to complete this task is given in the "Your task" section below.

   ### FASTA sequences

   The data we will be working with is in FASTA format. FASTA format is defined by two types of line: header lines which begin with ">" (i.e., the first character in the line **has to be** ">"), and sequence lines

which do not contain ">" characters. Each header line is associated with all sequence lines following it until another header line is reached. For example, in the following FASTA file, "header 1" has 3 lines of sequence, while "header 2" has 2 lines of sequence

```
>header 1
ATCGTCGCAAA
ATCGTGTGGTA
CCGTGTTG
>header 2
GGTGTGTGGTG
AGTAGTAA
```

Sequence lines can be any length, although a common practice is to restrict lines to 60 characters in length. (Consider what that would mean for searching the file using something like `grep`!)

FASTA is used to store DNA, RNA, and protein sequence. There is no metadata lines inside the file which tell you whether the data are DNA, RNA, or protein. However, some organizations have defined specifications to provide more information about the nature of sequences (see for example https://www.ncbi.nlm.nih.gov/genbank/fastaformat/).

Note that DNA and RNA is often double stranded. FASTA representations of DNA and RNA sequences only represent one of the two strands.

Instead of metadata within the file, file extensions are commonly used to identify the nature of sequences contained in a FASTA file. For example, generic file extensions like ".fasta" and ".fa" can be used for any FASTA sequences, but do not indicate the contents. ".fna" and ".faa" specify that the file contains **n**ucleic **a**cids (DNA/RNA) or **a**mino **a**cids (protein), respectively. As with other files, it is good practice to use informative file extensions to help to organize your data.

A common datatype that is stored in FASTA format is genome assemblies. Genome assemblies are hypotheses about the sequence of an organism's genome that was generated using sequencing data (e.g., Illumina sequencing reads). Sequence assembly software often produces genome assemblies with generic sequence headers. For example, Spades (https://github.com/ablab/spades) produces headers that contain length and coverage information, but nothing that links headers from the same assembly (e.g., ">NODE_1_length_1121141_cov_33.426083").

In many cases, bioinformaticians need to analyze large numbers of FASTA files at once. In those cases it is often useful to modify sequence headers to contain additional information, such as which sample they are associated with. Your task for this question is to write a Bash script that does that.

In this weeks assignment folder are two genome assemblies of the pathogenic bacterium *Pseudomonas aeruginosa*. Their filenames contain unique identifying information (their accession numbers in the European Nucleotide Archive).

Your task

Write a script that adds the accession number (i.e., the filename **without the extension**) to the beginning of each header line in the file and outputs the modified FASTA sequence to a new file. The output file must be a valid FASTA format (i.e., follow the format rules described above).

The usage of your script should be `change_headers.sh <input file> <output file>`

Hints:

- Don't forget that you can break a complex problem down into simpler steps by using a pipeline.
- `sed` can use environment variables in expressions when the expression is in *double* quotes.

```
bar="goodbye"
echo "hello there" | sed "s/hello/$bar/"
```

6. "find_perfect_matches.sh" (30 points)

For this question, you will analyze the outputs of the script you made in question 4.2.

One application in which you may want to analyze many assemblies at once is when searching for the presence of a sequence of interest. You may wish, for example, to determine which samples contain a gene (or allele) of interest.

The following sections provide some background on the BLAST tool and biological question we are using BLAST to address. If you are already familiar with running BLAST in the command line, all the information required to complete this task is given in the "Your task" section below.

BLAST

The most widely used program for searching a sequence database for matches to a query sequence is BLAST. We don't have time to go over the BLAST algorithm in detail in this course. However, I recommend you watch the short videos here, which offer an excellent description.

BLAST is able to rapidly search for approximate sequence matches between a query sequence and a subject sequence or database. You can search online databases hosted by the NCBI (like those you would search using the web application) or local sequences and databases.

The BLAST command line application allows you to control parameters of the algorithm as well as the output format of the results. For our purposes we are going to stick with the tab-delimited format, `-outfmt 6`. This format has twelve default fields as well as a number of optional fields. You can view all possible fields in the help message (note: BLAST does not follow the short and long option convention described in class. All options are single dash. Help is called with, e.g., `blastn -help`). Output fields can be specified as follows: `-outfmt '6 [std] <field> ...'`, where std can be included if you would like the default twelve fields output. For example, `-outfmt '6 sseq'` would output only one field - the matched sequence in the subject, while `-outfmt '6 std sseq'` would output the default twelve fields and a thriteenth field containing the matched sequence in the subject. You can find a full list of possible fields that can be included in outfmt 6 here

BLAST output when run with `-outfmt 6` is tab-delimited, with one sequence match reported on each line. Note that your BLAST results may wrap to multiple lines depending on your terminal size and text size. However, the representation in your terminal does not impact the organization of the output.

Another option to be aware of for this exercise is `-task <task>`. See the online documentation for a description of the various options https://www.ncbi.nlm.nih.gov/books/NBK569839/. We will be

searching for matches for a query sequence of 28 nucleotides in length. For short nucleotide sequences, `-task blastn-short` should be used.

As we are comparing nucleotide sequences here, `blastn` is the program we will use. The basic usage of `blastn` for our purposes is:

```
blastn -query <query fasta file> -subject <subject fasta file> -task <task>
- outfmt <desired format> [-out <outfile>]
```

Some things to note about the above usage:

- If no output file is specified, the BLAST results will be output to stdout.
- Only one query file and one subject file can be provided. If you want to blast multiple queries against multiple subjects, you must combine the corresponding files. All queries and subjects must be valid FASTA format.

CRISPR

To illustrate a case in which BLAST can be used, I have provided you with two assemblies ("ERR430992.fna", "ERR431227.fna") as well as another FASTA file ("CRISPR_1f.fna"). The "CRISPR_1f.fna" file contains a single, short sequence. This sequence is associated with a bacterial immune system called CRISPR.

CRISPR (Clustered Regularly Interspersed Short Palendromic Repeats) systems are present in about 40% of bacterial species. They are defined by loci containing repeat sequences that are evenly spaced, with intervening sequences (called spacers) that are typically homologous to viruses and plasmid sequences. The function of CRISPR systems as an immune system is acheived through sequence-specific targeting of CRISPR-associated (Cas) proteins to destroy invading viruses etc. Cas proteins typically achieve this destruction of invading genetic elements through DNA/RNA cutting activity. Spacers are the immune memory that provides the sequence specific targeting that guides the Cas proteins to destroy only specific non-self sequences.

In recent years, the sequence-specific cutting activity of the Cas proteins associated with CRISPR systems has been developed into genetic tools that allow the genetic editing of almost any organism! You may have already heard of CRISPR systems because of these advances. For the exercise at hand, we are going to focus on the bacterial immunity aspect of CRISPR systems, rather than the gene editing.

The most common type of CRISPR system encoded by *P. aeruginosa* is type I-F. If a strain of *P. aeruginosa* has CRISPR repeats and *cas* genes, then it has a (likely) active CRISPR immune system. The absence of either repeats or *cas* genes indicates no active CRISPR system. Your task here is to perform an analysis to determine which (if any) of the provided assemblies encodes type I-F CRISPR repeats.

Your task

Write a script that runs a BLAST of a query file against a subject file, writes **only perfect matches** to an output file, and prints the number of perfect matches to stdout.

The usage of your script should be `find_perfect_matches.sh <query file> <subject file> <output file>`

Hint: Think about what information you need to identify a perfect sequence match. In order for two sequences to be considered identical, they must have 100% sequence identity and equal length. Which output fields do you need to add (if any) to the BLAST output to determine which matches meet those criteria? Don't hard code a specific value for the length. Get it from the BLAST output.