Programming for Bioinformatics | BIOL 7200

Exercise 8

Submission specifications

1. Your module ***does*** need to be generalizable to any FNA input files. We're going to be using this code to analyze different files moving forward. Don't hard code anything to the provided example files
2. You may only use Python standard library modules
3. Your submission should consist of either a module file called "ispcr.py" or the contents of a package directory that includes an file called "__init__.py". If you submit an init file, all submitted python files will be moved to a directory called "ispcr" and treated as if you had submitted a package directory. Do not try to submit packages with nested structures of subpackages (i.e., directories) as gradescope can't handle directories. The autograder will tell you if it thinks you submitted a single module file or a multi-file package and whether it was able to import your code correctly
4. Do not upload any of the q[123].py files as part of your submission.

Assignment description

This assignment is to write a Python package that performs *in silico* PCR (isPCR). isPCR is the simulation of the Polymerase chain reaction (PCR) that is commonly used in wet-lab settings to amplify target DNA sequences. It uses the same concepts that underpin PCR, but uses computational approaches to simulate the outcome by modeling the chemistry involved in PCR.

This assignment will really show you how much the concepts we have covered in class can be combined to tackle bioinformatics problems. This course is most focussed on the coding part of bioinformatics. Therefore, rather than asking you to invent an approach to perform isPCR, this assignment is composed of three "questions", where each "question" is one of the steps in the isPCR process. If you answer all three questions correctly, you will have written a python package that can perform isPCR from start to finish. I will describe what each step should achieve and provide you with the following:

- The input that should be used for this step (i.e., the output from the previous step or the initial input files)
- A script that will import and run a function from your package (more details below)
- The expected output that you should see printed to your terminal when you run the provided script

Using a stepwise structure for this assignment will allow you to focus on applying the coding concepts in class to each individual problem. In addition, if you can't finish the first step, you won't lose all the points as you can still proceed with the remaining steps. Furthermore, breaking up the development of a complex program such as this is an effective development strategy so this exercise will practice that sort of development style in a way that previous exercises have not.

The way you will achieve the stepwise development process while also being able to submit solutions to each separate question is by taking advantage of the Python module system. All of the processing will be performed by functions you write within a package that you will call "ispcr". I have provided you with three scripts. One script for each step. Each of these scripts imports your package and runs a function from it. You need to write the called function to perform the appropriate processing in order to return the expected output. A good approach would be to write all the functionality in other functions within your module and then simply use the function my script calls to call the relevant functions you wrote.

You can organize your module/package however you like behind the scenes. For this assignment, the "public API" (the stuff users of your package like the autograder use) will just be the specified functions. How you implement the functionality behind the scenes is up to you. The one restriction on how you organize your code is that you can't use subpackages. Because gradescope doesn't offer a way for us to provide inline comments or easily review code if you were to upload a tarball of your package/module, you will have to upload files. I've set up the autograder to detect whether you are submitting a package based on whether you upload an __init__.py file, but while that works for a single directory, it would be too complicated to contrive a system to allow you to upload multiple directories and their associated __init__.py and module files. If you have any questions about what the autograder is expecting please ask!

This week's assignment will form the first part of a larger Python program. Over the remaining weeks we will add other components and together the parts will create an impressive magnum opus that uses many of the concepts we've covered in this course.

PCR background

Below is a brief description of Polymerase chain reaction (PCR). If you prefer videos with drawings or visual effects, you may find the following useful. First a video by Khan academy with a short discussion of the different amplification products you get early vs late in the process of PCR. Second, a short video series by Thermo Fisher: 1, 2, 3, and 4. Finally, the NIH has a nice, simple, silent video that just shows the basic process in terms of the amplification of DNA without any explanation.

For the purposes of performing isPCR, the following is a description of what you need to know about DNA and PCR.

DNA - nucleotides and synthesis

DNA typically exists in a double stranded state. Double stranded DNA is composed of two anti-parallel molecules of DNA. Anti-parallel means that the two molecules are parallel to one another, but that they are in opposite orientations. It is also common to describe each DNA molecule's sequence as being the "reverse complement" of that of the other strand.

The fact that the two DNA molecules are oriented opposite to one another is significant because of the chemical differences between the two ends of the molecules. Specifically, DNA is composed of a long chain of nucleotides, which each have a hydroxy group (an oxygen and a hydrogen atom - OH) on one end (the 3' or "three prime" end) and a phosphate group (one phosphorous and four oxygen atoms - PO_4) on the other end (the 5' or "five prime" end). Nucleotides join together in a chain by forming a covalent bond between the 3' group of one nucleotide and the 5' group of the next. Therefore, any linear DNA molecule will have a 3' end and a 5' end simply because it is composed of nucleotides with those ends.

Beyond simply providing a naming convention, the 3' and 5' ends of nucleotides are significant because of their roles in DNA synthesis. Forming a covalent bond between two nucleotides requires energy. This energy is provided by breaking an existing bond in the nucleotide being added to the DNA molecule. The bond being broken is a bond between two phosphate groups, i.e., the group on the 5' end of the nucleotide. Breaking this bond provides the energy to attach the 5' end of the incoming nucleotide to the 3' end of the nucleotide already present in the DNA molecule.

There are two consequences of this mechanism of DNA synthesis. First, as energy is required from removing part of the *incoming* nucleotide's 5' phosphate, DNA can only be extended on its 3' end. Secondly, as the phosphate group is lost during this process, the 5' end of DNA is effectively used up so it can't support DNA synthesis in the opposite direction. For these reasons, DNA synthesis is strictly directional.

As an aside, these phosphate-phosphate bonds are a common source of energy in cells. You've likely heard of ATP or adenosine triphosphate, which is a nucleotide. The energy that molecule provides is stored in its phosphate groups.

While the "backbone" of a DNA molecules is covalent bonds between nucleotides, the bonds that link two molecules of DNA into a double-stranded helix are much weaker "hydrogen bonds". Breaking a covalent bond requires quite a lot of energy. Hydrogen bonds are more transient and can break more easily. For example, heat will break hydrogen bonds.

When a genome is replicated during cell division, the two molecules of DNA in a double strand are pulled apart. Each strand is then copied starting at some point and proceeding in a directional manner as described above. This copying is possible because of the nature of the hydrogen bonds between nucleotides in different DNA molecules.

DNA is made up of four different nucleotides, which are typically abbreviated as A, T, C, and G. A and T can form compatible hydrogen bonds, as can C and G. However, neither A nor T form compatible hydrogen bonds with C or G. The compatibility of hydrogen bonds between nucleotides (also called "bases") is also called "complementary base pairing". This chemical affinity between the bases allows DNA to be copied by simply relying on the chemical properties of the template DNA sequence. A protein called "DNA polymerase" catalyses the synthesis of DNA. The videos linked above illustrate this process of DNA replication through the copying of a template strand.

PCR

PCR, or polymerase chain reaction, is pretty much what is sounds like. It is a chain reaction, in that it has an exponential progression, and it uses DNA polymerase. The general idea behind PCR is to use the natural DNA synthesis process to amplify (produce many copies of) a target region, or regions. Here, we'll briefly describe how that is achieved in a test tube as well as how we can simulate that process using a computer.

PCR is composed of three steps: melting, annealing, and extension. The first two steps are where the "targeting" of what we want to amplify is achieved, while the extension step is simply normal DNA synthesis.

In the above description of DNA synthesis, we only discussed DNA synthesis in the context of adding new nucleotides to an existing DNA molecule. So how does a brand new DNA molecule get started? Using primers. In normal DNA replication, an enzyme called a primase produces short RNA fragments which anneal to the template DNA molecule and "prime" or seed the synthesis of a new DNA molecule. In PCR, synthetic, short sequences of DNA (sometimes called "oligonucleotides" or "oligos") are used. In both cases, the short sequences anneal to the template strand according to the same complementary base pairing rules that govern DNA synthesis. Synthetic primers can therefore be designed to only anneal to a specific sequence by complementary base pairing. This leads to those primers only initiating the replication of the sequence next to their annealing site.

The three steps of PCR are therefore achieving the following results. The melting step breaks the hydrogen bonds between the double-stranded DNA. This is necessary because in order for the primers to anneal to their target sequence, they need to be able to access the nucleotides and form new hydrogen bonds with them. They can't access the nucleotides while the DNA is double stranded. Next, comes the annealing step. The annealing step is simply reducing the temperature of the test tube so that hydrogen bonds can form stably. In this step, some of the double-stranded DNA molecules will reform. However, as PCR reactions are typically performed with huge numbers of primer molecule, most DNA molecules will instead anneal to a primer. Once primer annealing has occurred, the temperature is again changed to the optimum temperature for the DNA polymerase to synthesize DNA. This is called the extension step. The whole process is then repeated as many times as desired, which typically depends on the amount of amplification is required.

During the first cycle (i.e., the sequence of temperature changes described in the last paragraph), the molecule of DNA being copied is often something massive like a chromosome. Therefore, the DNA polymerase might amplify far beyond the end of the target sequence. For that reason, two primers are used, with one primer on each end of the target region. Each primer will target sequence on one of the DNA molecules so that it will initiate synthesis in the opposite direction to that of the other primer. As each primer initialized replication at a fixed point, many new molecules of DNA will be made which start at that point. Once a primer anneals to a molecule of DNA replicated using the other primer, DNA synthesis of that molecule will only be able to proceed until it reaches the point that the other primer started the original synthesis. This part is very hard to describe with words. Instead, I recommend you check out the NIH video linked above (and also here for convenience), which shows what I am trying to describe.

The end product of performing many cycles of PCR is that wherever two primers annealed sufficiently close together and facing towards one another, there will now be many copies of the original sequence. For example, after 20 cycles, you would have 1,000,000 times as many copies of the original sequence (assuming 100% efficiency of amplification).

Specificity

Crucially, there is nothing special about PCR that means you will always get the amplicon (amplified sequence) that you wanted. *Any* region that has two primers anneal facing one another and sufficiently closely together will be amplified. The specificity of amplifying just a target sequence depends entirely on careful design of the primer sequence. It can be quite difficult to design primers if you want to amplify a repetitive region or one with high similarity to a region you don't want to amplify. Typically, the process involves trial and error of finding a sequence which has a high enough melting temperature that you can use an annealing temperature which will not allow off-target annealing to occur. There are online melting temperature calculators which will give you a sense of the experience. As an aside, melting and annealing temperatures are about the same. They are just the inverse processes, like freezing vs melting water, which happens at 0 degrees celsius.

The specificity of primers is an important feature to consider if you want to perform isPCR. As any primer annealing to any sequence is sufficient to initiate DNA synthesis, you need to consider all binding sites. If two primers anneal to an off-target location facing one another and sufficiently close together, they will produce a fully fledged amplicon. If you wanted to write something a bit more involved, you could let a user set the annealing temperature to use and you could calculate the melting temperature of all primer matches to see which ones would actually amplify. However, for our purposes, we're just going to use a flat cutoff.

isPCR

isPCR is simply using a computer to simulate what would happen if you performed a real PCR reaction with a certain set of primers and template sequence. As isPCR tries to replicate the results you would get in a real PCR, it follows the same rules that govern the results of a real PCR reaction.

isPCR is performed in three steps:

1. Identify locations where primers would anneal to the target sequence
2. Identify pairs of locations where two primers anneal close enough together and in the correct orientation for amplification to occur
3. Extract the amplified sequence

Each step is described in more detail in the relevant sections below. For each question, there is an instruction section and an expected output section which shows what the provided script should print to the terminal.

1. Identifying primer annealing location (30 points)

Instructions

The first step of PCR is primer annealing. In a test tube, the locations that primers anneal is determined by complementarity between the sequence of the primers and template sequence. Specifically, more hydrogen bonding leads to stronger attachment and a higher melting temperature. Mismatches reduce the strength with which a primer will anneal (as fewer hydrogen bonds are formed) and therefore reduce the melting temperature.

The way we can simulate primer annealing *in silico* is by identifying primer sequence matches within the template sequence which have fewer than some threshold of mismatches. We will use a percent identity cutoff here to keep things simple instead of calculating melting temperatures. BLAST is the go-to tool whenever you want to identify sequence matches so we'll use that here too.

For this "question" You need to begin writing your package. You should call your package "ispcr" and put it in the same directory as the q[123].py scripts. Within your package, you need to write code which is compatible with how the q1.py script is calling the ispcr package. Specifically, you should do the following:

1. Write functionality in your package to run BLASTN to search for sequence matches between the provided primer file and assembly file. Use `-task blastn-short` as primers are typically < 50bp long. You should run BLASTN within your package using the Python package "subprocess". Use `-outfmt '6 std qlen'`.
2. Process the output of your BLAST to keep full length hits with percent identity of at least 80%. You can use Python code to perform this filtering or can use `awk` with subprocess. Your choice.
3. Write a function called `step_one()` which will be run by q1.py. This function should perform all the necessary steps (or call functions that take those steps) to take the provided input and produce the expected output. the function signature of `step_one()` should look like the following

```
def step_one(primer_file: str, assembly_file: str) -> list[list[str]]:
```

It is up to you how you organize your package. You can make it a directory containing modules and an __init__.py, or you can make it a single module script. If you want, you can perform all of the steps within the function `step_one()`. however, I strongly recommend that you write separate functions to perform the different components of the processing and then just call those functions within `step_one()`. That way when you continue using this package in the next assignment, you will already have functions written in a more versatile form. You can also adjust the structure of your package after we review this assignment.

Note that the expected output below has all of the BLAST fields as `str` instances. You can convert the format of fields within your functions behind the scenes to `int` or `float` if you like. However, to satisfy the above function signature, you must convert everything back to a `str` before returning it.

Expected Output

```
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '46920', '46938',
'0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '47211', '47192',
'0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '144156', '144174',
'0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '144447', '144428',
'0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '149622', '149640',
'0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '149913', '149894',
'0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '401483', '401501',
'0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '401774', '401755',
'0.45', '26.1', '20']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2321911',
'2321930', '0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '2322202',
'2322184', '0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2683111',
'2683130', '0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '2683402',
'2683384', '0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2688655',
'2688674', '0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '2688946',
'2688928', '0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2694199',
'2694218', '0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '2694490',
'2694472', '0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2771804',
'2771823', '0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '2772095',
'2772077', '0.005', '32.4', '19']
['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2945147',
'2945166', '0.45', '26.1', '20']
['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '2945438',
'2945420', '0.005', '32.4', '19']
```

2. Identifying pairs of primer annealing sites which would yield an amplicon (40 points)

Instructions

In a test tube, amplification occurs wherever a primer has annealed. However, as I have tried to explain above (an as the videos have illustrated), amplification will only occur in an exponential manner in locations where two primers aneal close together and pointing towards one another.

Locations where only one primer anneals will still produce copies of the template. However, as only one strand will be copied each round, there will only be 1 copy of single stranded DNA produced per cycle (i.e., N copies of single stranded DNA, where N is the number of cycles of PCR performed). When two primers anneal close together and in the correct orientation, an extra copy of both strands will be formed. The strand formed by one primer can then act as the template for the other primer in the next cycle, and so on. This results in 2^N copies of that region being produced. Within only a small number of cycles, the production of single strands of DNA becomes negligible compared to the exponential amplification of properly amplified regions.

For isPCR, we can ignore the negligible amplification that occurs where only one primer binds. Instead, our simulation of PCR needs only take into account regions where both primers amplify. In order for such amplification to occur, there are two requirements that I have already stated several times:

1. Both primers need to anneal pointing towards one another. i.e., extension of each primer must proceed towards the annealing site of the other primer.
2. Both primers must anneal sufficiently close together.

We have discussed the first requirement quite a bit, but why do primers need to be close together? Basically, it is a consequence of the processivity of DNA polymerase as well as the quantity of dNTPs (nucleotides) added to the reaction and the depletion of those nucleotides.

Polymerase processivity is a trait that is determined by two things: affinity of the polymerase for DNA (i.e., how often does it fall off), and the rate with which the polymerase can incorporate new nucleotides into the grown DNA molecule. Typical polymerase processivity might be in the range of 500bp to 2kbp per minute. In a test tube, you can exert some influence over off-target amplification by controlling the extension time of your cycles. If your off-target amplicon is much longer than your desired amplicon, then you can reduce extension time to increase the proportion of target amplicon produced.

Nucleotide depletion is simply a consequence of adding nucleotides to the synthesized DNA molecules. At the start of a PCR reaction, some fixed amount of nucleotide mix is added. Every cycle, some (exponentially increasing) amount of nucleotide is used up during the synthesis of DNA. It is therefore not feasible to amplify arbitrarily long DNA sequences by PCR.

The maximum amplicon length you can typically achieve with PCR is a few kbp, depending on your polymerase. For this assignment, we're going to assume a maximum amplicon size of 2kbp. You should write your code to allow this to be controlled though. It would be a good thing to have command line control over in the finished product.

For this question you need to do the following:

1. Write functionality in your package to identify BLAST hits that are less than 2kbp apart and pointing towards one another. The input to this function will be the output of question 1. You can see exactly how your package will be used in the q2.py script.
2. Write a function called `step_two()` which will be run by q2.py. This function should perform all the necessary steps to take the provided input and produce the expected output. the function signature of `step_two()` should look like the following

```
def step_two(sorted_hits: list[str], max_amplicon_size: int) ->
list[tuple[list[str]]]:
```

Finally, a little more information about how to determine if two primers are "pointing towards one another". DNA sequence is, by convention, written in a manner that reflects its directional nature. Whenever you see DNA sequence, you are seeing the 5' base at the start and the 3' base at the end (if read left to right). Occasionally you will see this explicitly. For example, 5'-ATCGTGAC-3'.

As DNA is written 5' to 3', the position in a FASTA file also follows this rule. When you are dealing with BLAST output, which reports where in the query and subject FASTA files a match was found, you can infer the direction of the match by comparing the start and stop of the match.

Consider, for example, the first two hits in the question 1 expected output. If we take just their "qseqid", "sstart", and "send" columns we see the following:

```
515F    46920   46938
806R    47211   47192
```

The 5' end of primer 515F matched at position 46,920, while the 3' end matched position 46,938. The 3' end matched a higher index position in the reference sequence. Amplification will therefore proceed towards higher index positions in the subject sequence (i.e., towards the right of the FASTA file).

The 5' end of primer 806R matched at position 47,211, while the 3' end matched at position 47,192. The 3' end matched a lower index position in the reference sequence. Therefore, amplification will proceed towards lower index positions in the subject sequence (i.e., towards the left of the FASTA file).

The location where primer 515F matched is a lower index position than that matched by primer 806R.

Finally, the 3' end of each primer matched 254 bases apart.

Taken together, these data indicate that the first two BLAST hits would produce a valid PCR amplicon. If you apply the same criteria to all other pairs of primers, you will identify the other amplicons. Do remember though, that any pair of primers can make an amplicon (even a pair of the same primer), so you need to check all pairs of matches to see if they would produce an amplicon!

Expected Output

```
(['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '46920', '46938',
'0.005', '32.4', '19'], ['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1',
'20', '47211', '47192', '0.45', '26.1', '20'])
(['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '144156', '144174',
'0.005', '32.4', '19'], ['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1',
'20', '144447', '144428', '0.45', '26.1', '20'])
(['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '149622', '149640',
'0.005', '32.4', '19'], ['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1',
'20', '149913', '149894', '0.45', '26.1', '20'])
(['515F', 'NZ_CP028827.1', '89.474', '19', '2', '0', '1', '19', '401483', '401501',
'0.005', '32.4', '19'], ['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1',
'20', '401774', '401755', '0.45', '26.1', '20'])
```

```
(['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2321911',
'2321930', '0.45', '26.1', '20'], ['515F', 'NZ_CP028827.1', '89.474', '19', '2',
'0', '1', '19', '2322202', '2322184', '0.005', '32.4', '19'])
(['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2683111',
'2683130', '0.45', '26.1', '20'], ['515F', 'NZ_CP028827.1', '89.474', '19', '2',
'0', '1', '19', '2683402', '2683384', '0.005', '32.4', '19'])
(['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2688655',
'2688674', '0.45', '26.1', '20'], ['515F', 'NZ_CP028827.1', '89.474', '19', '2',
'0', '1', '19', '2688946', '2688928', '0.005', '32.4', '19'])
(['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2694199',
'2694218', '0.45', '26.1', '20'], ['515F', 'NZ_CP028827.1', '89.474', '19', '2',
'0', '1', '19', '2694490', '2694472', '0.005', '32.4', '19'])
(['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2771804',
'2771823', '0.45', '26.1', '20'], ['515F', 'NZ_CP028827.1', '89.474', '19', '2',
'0', '1', '19', '2772095', '2772077', '0.005', '32.4', '19'])
(['806R', 'NZ_CP028827.1', '85.000', '20', '3', '0', '1', '20', '2945147',
'2945166', '0.45', '26.1', '20'], ['515F', 'NZ_CP028827.1', '89.474', '19', '2',
'0', '1', '19', '2945438', '2945420', '0.005', '32.4', '19'])
```

3. Extracting amplified sequences (30 points)

Instructions

Finally, you can amplify the sequences produced by your PCR reaction. In the test tube, these amplicons are composed of double stranded DNA which includes both primers as well as all the intervening sequence. However, it is common to represent just the sequence between the two primers (excluding the primer sequence) as the amplicon sequence. Here, I'll describe how that will work in our isPCR program.

Having identified each pair of primer annealing sites which would be expected to result in an amplicon, the final step is to extract the sequences of those amplicons. This step simply involves describing the expected location of the amplicon in BED format. i.e., the contig, start, and stop position of each amplicon. The created BED format file would look something like

```
NZ_CP028827.1   46938   47191
NZ_CP028827.1   144174  144427
...
```

To perform the sequence extraction, you might be inclined to use the same approach as in exercise 7 where you read the assembly file into Python. However, while that would certainly work, this assignment is assessing your ability to use subprocess. Therefore, in this exercise you have to use subprocess to run `seqtk` to perform the extraction of amplicon sequences.

There are a few ways you could go about creating your `seqtk` command. Specifically, as your BED file is currently just a `str` object in your Python script, you will need to figure out a way to pass it to `seqtk` and get the desired output. `seqtk` expects a BED *file* so you will need to pass a filepath. You can achieve that by writing your BED formatted `str` to a file (perhaps consider the `mktemp` equivalent in Python - the "tempfile" package in the standard library). Alternatively, you might refresh your memory about how process substitution works in Bash and use an approach that utilizes that. Any approach that uses `seqtk` to extract the sequences is acceptable for this assignment so perhaps try both and stick with whichever you prefer.

seqtk returns its output to stdout. Your function should return that str output rather than writing it to a file directly.

The signature of step_three() should be as follows:

```
def step_three(hit_pairs: list[tuple[list[str]]], assembly_file: str) -> str:
```

Expected Output

```
>NZ_CP028827.1:46939-47191 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
TACGGAGGGTGCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGCATGCAGGTGGTTTGTTAAGTCAGATGTGAAAGCCCTGG
GCTCAACCTAGGAATCGCATTTGAAACTGACAAGCTAGAGTACTGTAGAGGGGGGTAGAATTTCAGGTGTAGCGGTGAAATGC
GTAGAGATCTGAAGGAATACCGGTGGCGAAGGCGGCCCCCTGGACAGATACTGACACTCAGATGCGAAAGCGTGGGGAGCAAA
CAGG
>NZ_CP028827.1:144175-144427 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
TACGGAGGGTGCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGCATGCAGGTGGTTTGTTAAGTCAGATGTGAAAGCCCTGG
GCTCAACCTAGGAATCGCATTTGAAACTGACAAGCTAGAGTACTGTAGAGGGGGGTAGAATTTCAGGTGTAGCGGTGAAATGC
GTAGAGATCTGAAGGAATACCGGTGGCGAAGGCGGCCCCCTGGACAGATACTGACACTCAGATGCGAAAGCGTGGGGAGCAAA
CAGG
>NZ_CP028827.1:149641-149893 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
TACGGAGGGTGCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGCATGCAGGTGGTTTGTTAAGTCAGATGTGAAAGCCCTGG
GCTCAACCTAGGAATCGCATTTGAAACTGACAAGCTAGAGTACTGTAGAGGGGGGTAGAATTTCAGGTGTAGCGGTGAAATGC
GTAGAGATCTGAAGGAATACCGGTGGCGAAGGCGGCCCCCTGGACAGATACTGACACTCAGATGCGAAAGCGTGGGGAGCAAA
CAGG
>NZ_CP028827.1:401502-401754 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
TACGGAGGGTGCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGCATGCAGGTGGTTTGTTAAGTCAGATGTGAAAGCCCTGG
GCTCAACCTAGGAATCGCATTTGAAACTGACAAGCTAGAGTACTGTAGAGGGGGGTAGAATTTCAGGTGTAGCGGTGAAATGC
GTAGAGATCTGAAGGAATACCGGTGGCGAAGGCGGCCCCCTGGACAGATACTGACACTCAGATGCGAAAGCGTGGGGAGCAAA
CAGG
>NZ_CP028827.1:2321931-2322183 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
CCTGTTTGCTCCCCACGCTTTCGCATCTGAGTGTCAGTATCTGTCCAGGGGGCCGCCTTCGCCACCGGTATTCCTTCAGATCT
CTACGCATTTCACCGCTACACCTGAAATTCTACCCCCCTCTACAGTACTCTAGCTTGTCAGTTTCAAATGCGATTCCTAGGTT
GAGCCCAGGGCTTTCACATCTGACTTAACAAACCACCTGCATGCGCTTTACGCCCAGTAATTCCGATTAACGCTTGCACCCTC
CGTA
>NZ_CP028827.1:2683131-2683383 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
CCTGTTTGCTCCCCACGCTTTCGCATCTGAGTGTCAGTATCTGTCCAGGGGGCCGCCTTCGCCACCGGTATTCCTTCAGATCT
CTACGCATTTCACCGCTACACCTGAAATTCTACCCCCCTCTACAGTACTCTAGCTTGTCAGTTTCAAATGCGATTCCTAGGTT
GAGCCCAGGGCTTTCACATCTGACTTAACAAACCACCTGCATGCGCTTTACGCCCAGTAATTCCGATTAACGCTTGCACCCTC
CGTA
>NZ_CP028827.1:2688675-2688927 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
CCTGTTTGCTCCCCACGCTTTCGCATCTGAGTGTCAGTATCTGTCCAGGGGGCCGCCTTCGCCACCGGTATTCCTTCAGATCT
CTACGCATTTCACCGCTACACCTGAAATTCTACCCCCCTCTACAGTACTCTAGCTTGTCAGTTTCAAATGCGATTCCTAGGTT
GAGCCCAGGGCTTTCACATCTGACTTAACAAACCACCTGCATGCGCTTTACGCCCAGTAATTCCGATTAACGCTTGCACCCTC
CGTA
```

```
>NZ_CP028827.1:2694219-2694471 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
CCTGTTTGCTCCCCACGCTTTCGCATCTGAGTGTCAGTATCTGTCCAGGGGGCCGCCTTCGCCACCGGTATTCCTTCAGATCT
CTACGCATTTCACCGCTACACCTGAAATTCTACCCCCCTCTACAGTACTCTAGCTTGTCAGTTTCAAATGCGATTCCTAGGTT
GAGCCCAGGGCTTTCACATCTGACTTAACAAACCACCTGCATGCGCTTTACGCCCAGTAATTCCGATTAACGCTTGCACCCTC
CGTA
>NZ_CP028827.1:2771824-2772076 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
CCTGTTTGCTCCCCACGCTTTCGCATCTGAGTGTCAGTATCTGTCCAGGGGGCCGCCTTCGCCACCGGTATTCCTTCAGATCT
CTACGCATTTCACCGCTACACCTGAAATTCTACCCCCCTCTACAGTACTCTAGCTTGTCAGTTTCAAATGCGATTCCTAGGTT
GAGCCCAGGGCTTTCACATCTGACTTAACAAACCACCTGCATGCGCTTTACGCCCAGTAATTCCGATTAACGCTTGCACCCTC
CGTA
>NZ_CP028827.1:2945167-2945419 Vibrio cholerae strain N16961 chromosome 1, complete
sequence
CCTGTTTGCTCCCCACGCTTTCGCATCTGAGTGTCAGTATCTGTCCAGGGGGCCGCCTTCGCCACCGGTATTCCTTCAGATCT
CTACGCATTTCACCGCTACACCTGAAATTCTACCCCCCTCTACAGTACTCTAGCTTGTCAGTTTCAAATGCGATTCCTAGGTT
GAGCCCAGGGCTTTCACATCTGACTTAACAAACCACCTGCATGCGCTTTACGCCCAGTAATTCCGATTAACGCTTGCACCCTC
CGTA
```