Coding session: Biopython for assembly analysis

**conda environment with required software**: BTG\_biopython

## **Overview**

* [Intro to practical](#_Intro_to_practical_1)

* [How to use Biopython within Spyder](#_How_to_use_1)

* [The practical: Write a python-script to filter a SPAdes assembly (fasta-file)](#_Task_2:_Write)
* [Bonus-exercises](#_Bonus_exercises)

## **Intro to practical**

Our preliminary inspection of the SPAdes assembly files revealed that they contain some small contigs which are not well supported by the reads (low k-mer coverage). This is very typical for SPAdes, so many bioinformaticians will have a python-script for filtering them off. It’s a very useful type of script to have, and not too difficult to write, so lets dive in!

To help you get started, we have written a template-script (filter\_contigs\_template.py) which you can use as a starting point. The template-file only needs a few very minor edits to work on your own computer. But it’s a very crude script, so we will work on how to improve it.

The template-script can also be used as a starting-point for other scripts working on fasta-files, we have some ideas for you in the bonus-exercise section.

Happy coding!

## **How to use Biopython within Spyder**

“Biopython” is an example of a “python module”, i.e. a kind of code library containing functions developed by others to make our lives a bit easier. “Biopython” contains code to read and write files commonly used with sequence data (fx. fasta-files and fastq-files), while making sure that the formatting is correct. It also contains functions for common things you may want to do with such files (fx. finding the reverse-complement of a sequence, or changing file-formats). There are loads of other python modules out there which you can (and should) use, so it’s important to know how.

To use the biopython module for reading and writing files, you should add a line like this to the top of your python script:

from Bio import SeqIO

For this to work at run-time, the module must be available to the python interpreter. If you are working from the command-line, you could activate a conda-environment containing biopython, i.e.

conda activate BTG\_biopython

Similarly, to use a conda-environment in Spyder, you must select it first. In Spyder this can be done as follows:

* Go to preferences
* Select “Python interpreter” in the sidebar menu
* Select “Use the following interpreter” and find your conda environment in the drop-down menu. If it is not available, write the full path, i.e.
	+ / home/gebt/micromamba/2025\_envs/BTG\_biopython/bin/python3
* Click “Apply”, and “OK”
* Go to the console menu, and select “Restart kernel”

You can verify that the environment was correctly loaded by running this line in Spyder (if it doesn’t complain, all is good):

from Bio import SeqIO

*Bonus-info*

The conda-environment we have made available to you for this practical was made as follows:

 conda create -n BTG\_biopython -c conda-forge spyder-kernels biopython

where the -n argument is the name we chose for the environment, and the -c argument specifies the conda channel in which the module can be found. Note that in addition to biopython the environment also includes “spyder-kernels” for interacting with spyder. Also note that it is important that the spyder-kernels in the environment match your spyder installation. If they don’t, SPYder will tell you that you need a different version when you re-start your kernel.

## **The practical: Write a python-script to filter a SPAdes assembly (fasta-file)**

Start by downloading the template-script for the practical from EVA (filter\_contigs\_template.py) and open it in SPYder. Check that the conda-environment containing Biopython has been selected as the python interpreter (see previous section). **We will take a look at the template-script in class first**.

Your first task will be to make the template-script work on your own assembly-file.

1. Modify the template python-script, so that it takes a SPAdes assembly-file of your own choice and filters it with a reasonable cut-off.

**Hint** You will need to change the variables “input\_fasta\_file”, “output\_fasta\_file” and the length cut-off.

The script works, but it can become much more useful if we write it a little smarter. Here are some suggestions.

1. Automate the generation of the output-file, so it gets named according to the input-file. For example, if your input-file is called “contigs.fasta”, you could name the output-file “contigs\_filt.fasta”.

**Hint**: This could be done by replacing the string “.fasta” with “\_filt.fasta” in the variable “input\_fasta\_file”.

1. Let the input fasta-file be specified as a command-line argument for the script (instead of writing the full path to the input-file inside the script).

**Hint**: You can use the sys-module for this, check your python hand-outs!

**Tip**: You can run your python-script with a command-line argument in Spyder by typing run <name\_of\_script> <input> in the console window

1. Modify the script so it can filter the contigs based on both length and k-mer coverage.

**Hint**: the k-mer coverage (and contig-length) is provided in the fasta-header. You could split the header into a list and extract the value corresponding to kmer-coverage.

## Bonus exercises

A lot more can be done, based on the template you have been working on for this practical.

### Assembly summary

Write a script that calculates number of contigs and total assembly length. It can be done as a minor variation of the script you have already written. Instead of writing your list of filtered contigs to a file, you could generate a summary of them.

Copy your script to a new file and remove the code for writing output. Modify the for loop that iterates over the filtered contigs and think of some summary stats you would like to collect, fx.

* Number of contigs larger than 500bp
* Total assembly length (in bp)
* Size of biggest contig

Think about writing some nice print statements for the user, so she/he knows what has been calculated!

### Cut out a contig from your assembly file

Perhaps there is a contig that looks particularly interesting? A putative plasmid that you would like to blast? You can write a python script that takes an assembly file and a contig-id as input (perhaps command-line arguments), and writes the specific contig to a new fasta-file