ACADEMIS

# Basics of Biopython

tutorial

# Table of Contents

# Biopython Tutorial for Python 3

Course material by Dr. Kristian Rother

with contributions by Allegra Via, Magdalena Rother and Olga Sheshukova.

## What is Biopython?

Biopython is a Python library for reading and writing many common biological data formats. It contains some functionality to perform calculations, in particular on 3D structures.

**The library and documentation** can be found at www.biopython.org.

## License

This tutorial is distributed under the conditions of the **Creative Commons Attribution Share-alike License 4.0** (CC-BY-SA 4.0).

# First Steps in Biopython

## Preparations

1. Start a Python console
2. Type `import Bio`

---

## 1. Load a sequence file

Load the FASTA file `ap006852.fasta` into Biopython.

```
from Bio import SeqIO

records = list(SeqIO.parse("ap006852.fasta", "fasta"))
dna = records[0]

print(dna.name)
print(dna.description)
print(dna.seq[:100])
```

Check whether the following statements are `True` or `False`:

- The command `print(len(dna))` displays the length of the sequence.
- Replacing `records[0]` by `records[1]` results in a different sequence record.
- Replacing `dna.seq[:100]` by `dna.seq[50:100]` displays a different portion of the sequence.

---

## 2. Manipulating a sequence

```
from Bio.Seq import Seq
dna = Seq("ACGTTGCAC")
print(dna)

result = dna._____()
print(result)
if result == 'GTGCAACGT':
    print('OK')
```

**Which of the following needs to be inserted to obtain `GTGCAACGT` ?**

1. reverse_complement
2. transcribe
3. translate

---

# 3. Calculating GC-content

Look up *Section 3.2* of the Biopython documentation on (http://biopython.org/DIST/docs/tutorial/Tutorial.html) to find out how to calculate the GC-content of a sequence.

What is the GC-content of the sequence loaded in task 1?

- 55.556
- 46.875
- 33.514
- 50.000

---

# 4. Print annotation of a GenBank file

Load the GenBank file `ap006852.gbk`. In contrast to a FastA file, this one contains not only the sequence, but a rich set of annotations. Load the file as follows:

```
records = list(SeqIO.parse("ap006852.gbk", "genbank"))
dna = records[0]
```

Answer the following questions:

### 4.1 Which command gives the species the sequence is from?

- `print(dna.annotation['species'])`
- `print(dna.annotations['organism'])`

### 4.2 Which command produces the bigger ID number (GenBank ID or PubMed ID)

- `print(dna.annotations['gi'])`
- `print(dna.annotations['references'][0])`

### 4.3 How can you view a list of available annotation fields?

- `print(dna.annotations.get('keys'))`
- `print(dna.annotations.keys())`

---

# 5. Count atoms in a PDB structure

The following code reads the 3D structure of a tRNA molecule from the file `1ehz.pdb` and counts the number of atoms.

**There is a bug in the program. Execute the program. Identify the problem and fix it.**

```
from Bio import PDB

parser = PDB.PDBParser()
struc = parser.get_structure("tRNA", "1ehz.pdb")

n_atoms = 0
for model in struc:
    for chain in model:
        for residue in chain:
            for atom in residue:
                print(residue.resname, residue.id)
            n_atoms += 1

print(n_atoms)
```

---

# 6. Retrieve entries from NCBI databases

Use the following code to download identifiers (with the `esearch` web app) and protein sequences for these identifiers (with the `efetch` web app) from the NCBI databases.

**The order of lines got messed up! Please sort the lines to make the code work.**

```
identifiers = records['IdList']

from Bio import Entrez

records = handle.read()

handle = Entrez.efetch(db="protein", id=identifiers, rettype="fasta", retmode="text")

open("globins.fasta", "w").write(records)

searchresult = Entrez.esearch(db="protein", term="hemoglobin", retmax=5)

records = Entrez.read(searchresult)

Entrez.email = "krother@academis.eu"
```

# Using NCBI E-utilities

# Using Entrez from Biopython

## Step 1: import Entrez

```
from Bio import Entrez
```

## Step 2: enter your e-mail

The NCBI server might block anonymous requests, especially big ones!

```
Entrez.email = "my@email.eu"
```

## Step 3: Call esearch to find IDs

```
handle = Entrez.esearch(db="value", term="keywords", retmax=100)
```

Parameters include:

| parameter | examples |
|---|---|
| db | nucleotide |
| | protein |
| | pubmed |
| term | human[Organism] |
| | hemoglobin |
| | hemoglobin AND alpha |
| retmax | 10 (identifiers returned) |

## Step 4: get a list of IDs out of esearch

```
records = Entrez.read(handle)
identifiers = records['IdList']
```

### Step 5: use efetch to retrieve entries

We use the list of identifiers from step 4:

```
handle = Entrez.efetch(db="value", id=identifiers, retmax="200",
         rettype="fasta", retmode="text")
records = Entrez.read(handle)
```

In addition to the above, parameters include:

| parameter | examples |
|---|---|
| id | single id |
| rettype | fasta |
| | gb |
| retmode | text |
| | xml |

## Documentation:

You find a full list of available options on http://www.ncbi.nlm.nih.gov/books/NBK25500/

## Example URLs

### 1. Searching for papers in PubMed

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=thermophilic,packing&rettype=uilist

### 2. Retrieving publication records in Medline format

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&id=11748933,11700088&retmode=text&rettype=medline

### 3. Searching for protein database entries by keywords

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=protein&term=cancer+AND+human

## 4. Retrieving protein database entries in FASTA format

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=protein&id=1234567&rettype=fasta

## 5. Retrieving protein database entries in Genbank format

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=protein&id=1234567&rettype=gb

## 6. Retrieving nucleotide database entries

http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?
db=nucleotide&id=9790228&rettype=gb

# Project: Diagnosing Sickle Cell Anemia

**Programming tasks with Biopython**

# Goal

Your goal is to develop an experimental test that reveals whether a patient suffers from the hereditary disease sickle cell anemia. The test for diagnosis should use a restriction enzyme on a patients' DNA sample. For the test to work, you need to know exactly what genetic difference to test against. In this tutorial, you will use Biopython to find out.

The idea is to compare DNA and protein sequences of sickle cell and healthy globin, and to try out different restriction enzymes on them.

This tutorial consists of four parts:

1. Use the module `Bio.Entrez` to retrieve DNA and protein sequences from NCBI databases.
2. Use the module `Bio.SeqIO` to read, write, and filter information in sequence files.
3. Use the modules `Bio.Seq` and `Bio.SeqRecord` to extract exons, transcribe and translate them to protein sequences.
4. Use the module re to identify restriction sites.

**Have fun!**

---

## What is sickle cell anemia?

At the beginning of the course, watch the 5-minute movie **"Sickle Cell Anemia"** by Paulo César Naoum and Alia F. M. Naoum: **http://www.youtube.com/watch?v=R4-c3hUhhyc**.

---

# 1. Retrieving DNA and protein sequences with Bio.Entrez

## 1.1 Search identifiers on NCBI

Write Python code that searches for the cDNA sequence of the sickle cell globin protein from NCBI. Use the `Entrez.esearch` function. As keywords, use 'sickle cell AND human NOT chromosome'. Print the resulting database identifiers (not the full sequences).

Use the NCBI examples from the back of this tutorial.

## 1.2 Retrieve sequences using identifiers

Use the identifiers from task 1.1, retrieve the full sequence with the `Entrez.efetch` function from the NCBI server. The parameter `rettype` should be 'fasta'.

Print the identifier and defline for each entry using a for loop.

## 1.3 Retrieve a single GenBank entry

In the output of task 1.2, locate the cDNA of the sickle cell globin *manually*. Copy the identifier. Use the identifier to download the full GenBank entry only for that sequence with `efetch`. Print the entry. The parameter `rettype` should be 'gb'.

## 1.4 Write an output file

Save the GenBank entry from task 1.3 to a file 'sickle.gb'.

## 1.5 Retrieve and write multiple GenBank entries

Combine `esearch` and `efetch` to retrieve entries for the gene sequences of the human globin family. Find appropriate keywords to limit the search to beta-globin and only complete coding sequences. Write the outcome to a file.

## Hint:

It is often more convenient to design the query in a browser window before moving to Python.

## 1.6 Optional Exercises

- Save the retrieved entries to a single FASTA file.
- Save each of the beta-globin sequences to a separate GenBank file.
- Use Entrez to search 100 recent references related to malaria and sickle cell anemia on PubMed.

# 2. Bio.SeqIO

**Reading, writing, and filtering sequence files**

## 2.1 Read a GenBank file

Read the ‘sickle.gb’ file from task 1.4 using the `SeqIO.parse()` function:

```
from Bio import SeqIO

records = SeqIO.parse(filename, format)
```

The first parameter of `parse()` is the filename, the format is ‘genbank’. Print the `records` object.

Find out how to see the actual entries.

## 2.2 Print information for one sequence

Use the `dir()` function on a single record object to find out what attributes it has. Print the id, name and description of the sickle cell globin entry.

## 2.3 Write a FASTA file

Save the GenBank entry from task 2.1 to a FASTA file using the `SeqIO.write()` function:

```
SeqIO.write(records, file, format)
```

The first parameter of `write()` is a list of sequence records, the second a file open for writing, and the third should be ‘fasta’.

## 2.4 Print information for multiple sequences

Print the id, name, and description of all human beta-globins.

## Hint:

This is a great occasion to exercise string formatting, e.g. to obtain tabular outpus:

```
print("{:10s} {:7d}".format('Ada', 33))
```

## 2.5 Filtering sequence entries

Print the same information as in task 2.4, but do not show non-globin entries. If the description contains either ‘vector’ or ‘isolate’, don’t print anything.

## 2.6 Optional Exercises

- collected the entries matching all criteria in a new list
- save the filtered list to a FASTA file
- Filter the list of sequence entries even further using your own criteria

---

# 3. Bio.Seq and Bio.SeqRecord

**Working with sequences**

## 3.1 The DNA sequence

Read the sequence of the sickle cell globin cDNA. Print the DNA sequence. Use the `dir()` function to find out the name of the attribute.

## 3.2 Transcribe DNA to RNA

Transcribe the sickle cell cDNA sequence to RNA and print it. Use the `transcribe()` method of a `Seq` object.

## 3.3 Translate RNA to protein

Translate the sickle cell RNA to a protein sequence and print it. Use the `translate()` method of a `Seq` object.

Save the protein sequence to a separate file.

## 3.4 Analyze annotations of beta-globin

In the file with many globins, find the beta-globin entry with accession `L26462`. Use the field `r.annotations['accessions']` on a `SeqRecord` object to find it. Write the entry to a separate GenBank file.

## 3.5 Extract sequence features

Print all features of the L26452 entry. Use the field `r.features` on a `SeqRecord` object.

## 3.6 Extract exons

Print all exon features and their attributes `start`, `end` and `nofuzzy_start`, `nofuzzy_end`:

```
for feature in seq['features']:
    print(feature)
    ...
    # add printing attributes here
```

For each exon, extract the interval

```
feature[nofuzzy_start:nofuzzy_end]
```

Use the indices to extract portions of the complete sequence. Concatenate all exon sequences to a single string.

## 3.7 Extract the beta-globin protein sequence

Transcribe and translate the concatenated exon sequence and print it.

## 3.8 Results

Print the sickle cell and healthy beta-globin sequence in subsequent lines or a text file. Also print the two corresponding protein sequences in subsequent lines. What differences do you see?

## 3.9 Optional Exercises

- What are the ‘N’ characters in the sickle cell globin cDNA? How did they affect the transcription/translation?
- Check in the documentation how to read an alignment of multiple sequences using Biopython
- Propose a PCR primer that specifically recognizes the sickle cell gene.

---

# 4. Pattern Matching

**Identification of restriction sites**

## 4.1 Familiarize with Regular Expressions

Do the first 3-4 exercises on the RegexOne website (http://www.regexone.com)

## 4.2 Search for a start codon

Use the `re.search()` function to locate the start codon (ATG) in the cDNA sequence of healthy beta-globin. The first parameter is a search pattern string, and the second is the string to be searched:

```
import re

match = re.search('ATG', sequence)
if match:
    print(match.start, match.stop)
```

## 4.3 Search for a restriction site

Create a regular expression using the `re.compile()` function for the restriction enzyme DdeI (cuts at `NN^CTNAG`). Search with a regular expression in both sickle cell and beta-globin DNA sequences.

For simplicity copy-paste both sequences to string variables in your program.

If the search method returns a match, print the start and stop found in both DNA sequences.

## 4.4 Try more restriction enzymes

Test patterns for the restriction sites of:

```
HinfI (GTNNAC)
BceAI (ACGGCNNNNNNNNNNNN)
BseRI (GAGGAGNNNNNNNNN)
EcoRI (GAATTC)
MstII (CCTNAGG)
```

on both DNA sequences. Which restriction enzyme could you use to specifically identify carriers of the sickle cell anemia gene?

## Optional Exercises

- Take a look at the website Regex101
- To facilitate the restriction analysis, replace the N's in the sickle cell DNA by the

corresponding positions from the healthy DNA. Print the resulting DNA sequence.

# Python BLAST Tutorial

## Overview

In this tutorial, you will automate BLAST queries with Python. You will learn how to run BLAST **locally**, **multiple times**, and how to **read BLAST results with Python**. In the process, you will build a **program pipeline**, a concept useful in many biological analyses independent of BLAST.

The tutorial consists of six parts:

1. Preparations
2. Running local BLAST manually
3. Running local BLAST from Python
4. Running BLAST many times with Python
5. Reading BLAST output with Biopython
6. Plotting the results

---

## Case Study: Plasmodium falciparum

We have the hypothesis that *Plasmodium falciparum* has adapted to the human organism during its long history as a parasite. Specifically, we want to examine whether proteins from *Plasmodium* are more similar to human proteins than one would expect. If this is true, we could interpret e.g. that more similar proteins help *Plasmodium* to evade the human immune system.

As a small sample study, we will BLAST a set of peptides from a few *Homo sapiens* proteins against the proteome of *Plasmodium falciparum*. As a control, we will use the proteome of *Schizosaccharomyces pombe*.

---

## 1. Preparations

### 1.1 Check whether BLAST+ is properly installed

Enter the two following commands in a Linux console:

```
makeblastdb
blastp
```

Both should result in an error message other than *command not found*.

## 1.2 Create a BLAST database for Plasmodium falciparum

Create a BLAST database for the *Plasmodium* proteins. First, open a console and go to the folder *data/* . Type:

```
makeblastdb -in Plasmodium_falciparum.fasta -dbtype prot
```

You should see a message similar to:

```
Adding sequences from FASTA; added 5414 sequences in 0.56993 seconds.
```

## 1.3 Create a BLAST database for the control organism

Please create a BLAST database for *Schizosaccharomyces pombe* as well.

## 1.4 Questions

- What files have appeared in the *data/* directory?
- Why do we need to create a database first? Why can't BLAST do that right before each query?

---

# 2. Running local BLAST manually

Before running a large series of BLAST experiments, we will run a small sequence as a technical proof of concept. We are using a sequence copied from the *Plasmodium* sequences, so we know that BLAST should generate a 100% match.

## 2.1 Create a query file

Create an empty file *query.seq* in a text editor. Write the following peptide sequence into the file:

```
DAAITAALNANAVK
```

Make sure that there are no other characters in the file (no empty lines or FASTA deflines). Save the file to the *data/* directory.

## 2.2 Running local BLAST against Plasmodium

Go to a console in the *data/* directory and type:

```
blastp -query query.seq -db Plasmodium_falciparum.fasta -out output.txt -outfmt 7
```

## 2.3 Running local BLAST against the control group

Repeat the above query for *Schizosaccharomyces pombe*.

## 2.4 Adjust output formats

Insert different numbers (1-7) for the **outfmt** parameter and re-run the query.

## 2.5 Questions

- Take a look at the BLAST output. Is the result what you would expect?
- Does the control group support your assumptions so far?
- Which of the output formats do you find the easiest to read?
- Which of the output formats is probably the easiest to read for a program?

---

# 3. Running local BLAST from Python

Now we are going to do exactly the same operation from a Python program. For this we will need the **os module**.

## 3.1 Introduction to the os module

Open the document *pipelines/os_module_puzzle.pdf*. Do the exercise.

## 3.2 Running BLAST from Python

Now we will use the function **os.system** to run BLAST. Create a Python script **run_blast.py** in the *data/* directory. Write the following commands into it:

```
import os

cmd = "blastp -query query.seq -db Plasmodium_falciparum.fasta -out output.txt -outfmt
 7"
os.system(cmd)
```

Execute the program.

## 3.3 Customizing the query

In order to make the BLAST command in Python more flexible, we will combine it from variables. Change the code to the following:

```
db = "Plasmodium_falciparum.fasta"
cmd = "blastp -query query.seq -db " + db + " -out output.txt -outfmt 7"
```

## 3.4 More variables

Now add separate variables for the query and the output file name as well.

## 3.5 Additional examples for using os

In the *pipelines/* directory you find more examples using the *os module*. If you like, try them out as well.

## 3.6 Questions

- Is the output of the Python BLAST run identical to the one you did manually? How can you check that?

---

# 4. Running BLAST many times with Python

## 4.1 Creating query files

The file *data/human_peptide.fasta* contains about 2000 peptides. We want to run BLAST for each of them. To do so, we need to write each peptide to a separate file.

First, create a new folder for the query files:

```
mkdir data/queries
```

The Python script **multiblast/split_fasta.py** does that using **Bio.SeqIO**. You can use it by typing in the *multiblast/* directory:

```
python split_fasta.py ../data/human_peptide.fasta ../data/queries
```

If you want, you can try writing that script by yourself.

## 4.2 Validate the queries

Make sure that the query files have been generated and that they are not empty. You can check both with:

```
ls -l data/queries
more data/queries/9568103_99.fasta
```

## 4.3 Create output directories

Prepare a place where the results from each BLAST run will be stored:

```
mkdir data/Plasmodium_out
mkdir data/Pombe_out
```

## 4.4 Run BLAST

You can run BLAST for all queries with the script **multiblast/run_blast.py**. It uses **os** for three different things:

1. Reading directory names as command-line parameters
2. Looping through all files in a directory
3. Running the BLAST command

**However, the program is incomplete.**

You need to complete the BLAST command inserting the file names from the given variables. Use the parameter *-outfmt 5* in order to create XML output. We will need this later to read it from Biopython.

When everything is done, you should be able to execute the script with:

```
python run_blast.py ../data/queries/ ../data/Plasmodium_falciparum.fasta ../data/Plasm
odium_out/
```

Inspect the result.

---

# 5. Reading BLAST output with Biopython

## 5.1 Reading XML data

Run the program **BLAST_XML/parse_blast_xml.py**.

```
python parse_blast_xml.py
```

## 5.2 Read one of your BLAST result files

Adjust the program to read one of your BLAST output files. Try to figure out how many HSPs there are, and how many are below an e-value threshold of 0.001.

## 5.3 Read all of your BLAST result files

Customize the program to read **all** of your result files. How many hits do you have in total. What is the hit with the highest score?

---

# References

BLAST+ is a new, faster (C++ based) version that replaces BLAST2, as of Oct 2013. Also see: http://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download

# Biopython Examples

## 1. Getting started

```
import Bio
from Bio.Seq import Seq
dna = Seq("ACGTTGCAC")
print(dna)
```

### (alternative)

```
from Bio.Alphabet import IUPAC
dna = Seq("AGTACACTGGT", IUPAC.unambiguous_dna)
```

## 2. Reverse complement, transcribing & translating

```
dna.reverse_complement()
rna = dna.transcribe()
rna.translate()
```

### (alternative)

```
from Bio.Seq import reverse_complement, transcribe, translate
reverse_complement("GCTGTTATGGGTCGTTGGAAGGGTGGTCGTGCT")
```

## 3. Calculating GC-content

```
from Bio.SeqUtils import GC
GC(dna)
```

## 4. Caculating molecular weight (DNA only)

```
from Bio.SeqUtils import molecular_weight
molecular_weight("ACCCGT")
```

# 5. Loading sequences from a FASTA file

```
from Bio import SeqIO
for record in SeqIO.parse("ls_orchid.fasta", "fasta"):
    print record.seq, len(record.seq)
```

# 6. Plotting a histogram of seq lengths with pylab

`pylab` aka `matplotlib` needs to be installed separately.

```
import pylab
sizes=[len(r.seq) for r in SeqIO.parse("ls_orchid.fasta","fasta")]
pylab.hist(sizes, bins=20)
pylab.title("%i orchid sequences\nLengths %i to %i" \
            % (len(sizes), min(sizes), max(sizes)))
pylab.xlabel("Sequence length (bp)")
pylab.ylabel("Count")
pylab.show()
```

# Authors

© 2015 Kristian Rother (krother@academis.eu)

This document contains contributions by Allegra Via, Magdalena Rother and Olga Sheshukova. I would like to thank Pedro Fernandes, Janick Mathys, Janusz M. Bujnicki and Artur Jarmolowski for their support during the courses in which the material was developed.

# License

Distributed under the conditions of a Creative Commons Attribution Share-alike License 3.0.