Python for biologists

Tutorial 3 - Functions and building python programs (on example of DNA data operations)

by Tobias Andermann

In this tutorial you will produce a python program which reads a DNA sequence alignment, changes the format of this alignment (into a user-provided format), and writes it to a new alignment file. You will be able to call this program from the bash command line and can thus easily up-scale it by looping over many alignment files. In case you don't work with DNA data, you can also write a different program that may be of more use for the data types you are more commonly working with.

1. Install Biopython

Use the pip installer to install biopython, which is a collection of useful functions surrounding DNA sequences and other realted objects. For this you can just type the following in your **bash command line** (not into your python editor).

```
pip install biopython
```

2. Read alignment

Read the alignment into python, using the `read()` function of Biopython:

In [1]:
```
from Bio import AlignIO
alignment = AlignIO.read(open("../data/primates_mtDNA.nex"), "nexus")
print(alignment)
```

```
IUPACAmbiguousDNA() alignment with 12 rows and 898 columns
YYYRRRSATAGGAGCAACCATTCTAATAATCGCACATGGCCTTA...CTT Lemur_catta
AAGCTTCACCGGCSRYSRYSRYSRYSRYSRYCACGGGCTTA...CTT Homo_sapiens
AAGCTTCACCGGCGCAATTATCCTCATAATCGCCCACGGACTTA...CTT Pan
AAGCTTCACCGGCGCAGTTGTTCTTATAATTGCCCACGGACTTA...CTT Gorilla
AAGCTTCACCGGCGCAACCACCCTCATGATTGCCCATGGACTCA...CTT Pongo
AAGCYRYRRYGGTGCAACCGTCCTCATAATCGCCCACGGACTAA...CTT Hylobates
AAGCTTTTCCGGCGCAACCATCCTTATGATCGCTCACGGACTCA...CTT Macaca_fuscata
AAGCTTTTCTGGCGCAACCATCCTCATGATTGCTCACGGACTCA...CTT M._mulatta
AAGCTTCTCCGGCGCAACCACCCTTATAATCGCCCACGGGCTCA...CTT M._fascicularis
AAGCTYYYYYYYYYYAACTATCCTTATAGTTGCCCATGGACTCA...CTT M._sylvanus
AAGCTTCACCGGCGCAATGATCCTAATAATCGCTCACGGGTTTA...CTT Saimiri_sciureus
AAGTTTCATTGGAGCCACCACTCTTATAATTGCCCATGGCCTCA...CTT Tarsius_syrichta
```

Looking at this overview of the alignment, we can see that there are some funny characters int he alignment (IUPAC ambiguity characters `Y,R,S`). We will learn later on how to replace these characters.

3. Inspect the alignment

Get the alignment length:

In [2]:
```
alignment.get_alignment_length()
```

Out[2]:
```
898
```

Print the first 10 characters of each sequence:

In [3]:
```
for record in alignment:
    print(record.seq[:10] + " " + record.id)
```

```
YYYRRRSATA Lemur_catta
AAGCTTCACC Homo_sapiens
AAGCTTCACC Pan
AAGCTTCACC Gorilla
AAGCTTCACC Pongo
AAGCYRYRRY Hylobates
AAGCTTTTCC Macaca_fuscata
AAGCTTTTCT M._mulatta
AAGCTTCTCC M._fascicularis
AAGCTYYYYY M._sylvanus
AAGCTTCACC Saimiri_sciureus
AAGTTTCATT Tarsius_syrichta
```

4. Modify sequences

Select one sample from the alignment:

In [4]:
```
sample_0 = alignment[0]
sample_0
```

Out[4]:
```
SeqRecord(seq=Seq('YYYRRRSATAGGAGCAACCATTCTAATAATCGCACATGGCCTTACATCATCCAT...CTT
', IUPACAmbiguousDNA()), id='Lemur_catta', name='Lemur_catta', description='', d
bxrefs=[])
```

You can extract the name and the sequence of the sample using the `.seq` and the `.name` function respectively.

In [5]:
```
sequence_0 = sample_0.seq
name_0 = sample_0.name
print(sequence_0)
```

```
YYYRRRSATAGGAGCAACCATTCTAATAATCGCACATGGCCTTACATCATCCATATTATTCTGTCTAGCCAACTCTAACT
ACGAACGAATCCATAGCCGTACAATACTACTAGCACGAGGGATCCAAACCATTCTCCCTCTTATAGCCACCTGATGACTA
CTCGCCAGCCTAACTAACCTAGCCCTACCCACCTCTATCAATTTAATTGGCGAACTATTCGTCACTATAGCATCCTTCTC
ATGATCAAACATTACAATTATCTTAATAGGCTTAAATATGCTCATCACCGCTCTCTATTCCCTCTATATATTAACTACTA
CACAACGAGGAAAACTCACATATCATTCGCACAACCTAAACCCATCCTTTACACGAGAAAACACCCTTATATCCATACAC
ATACTCCCCCTTCTCCTATTTACCTTAAACCCCAAAATTATTCTAGGACCCACGTACTGTAAATATAGTTTAAA-AAAAC
ACTAGATTGTGAATCCAGAAATAGAAGCTCAAAC-CTTCTTATTTACCGAGAAAGTAATGTATGAACTGCTAACTCTGCA
CTCCGTATATAAAAATACGGCTATCTCAACTTTTAAAGGATAGAAGTAATCCATTGGCCTTAGGAGCCAAAAA-ATTGGT
GCAACTCCAAATAAAAGTAATAAATCTATTATCCTCTTTCACCCTTGTCACACTGATTATCCTAACTTTACCTATCATTA
TAAACGTTACAAACATATACAAAAACTACCCCTATGCACCATACGTAAAATCTTCTATTGCATGTGCCTTCATCACTAGC
CTCATCCCAACTATATTATTTATCTCCTCAGGACAAGAAACAATCATTTCCAACTGACATTGAATAACAATCCAAACCCT
AAAACTATCTATTAGCTT
```

Now replace all invalid characters (not A,C,T,G,a,c,t,g,-) with `N` using the `re.sub()` function that you learned in the intro tutorial. This function can only be applied to string objects, but our sequence is in the biopython specific format called `Bio.Seq.Seq` . You can check the format of any variable with the `type()` function.

In [6]:
```
type(sequence_0)
```

Out[6]:
```
Bio.Seq.Seq
```

If you want to convert the sequence into string format, you can use the `str()` function:

In [7]:
```
sequence_string = str(sequence_0)
```

Task: Replace all characters in the sequence (`sequence_string`) that are **not** A,C,T,G,a,c,t,g,- with `N` and save it under the new variable `new_sequence_string`. The string should look as the one shown below:

Tip: Use the `re.sub()` function we learned in the intro-tutorial. Remember how you can use the `^` character to say: Replace everything except XXXXX . The command should look as follows, where you replace `XXXXX` with the characters you **do not want to be replaced** and `Y` is the character you want to replace all remaining characters with.

```
re.sub('[^XXXXX]','Y',string)
```

In [15]:
```
print(new_sequence_string)
```

```
NNNNNNNATAGGAGCAACCATTCTAATAATCGCACATGGCCTTACATCATCCATATTATTCTGTCTAGCCAACTCTAACT
ACGAACGAATCCATAGCCGTACAATACTACTAGCACGAGGGATCCAAACCATTCTCCCTCTTATAGCCACCTGATGACTA
CTCGCCAGCCTAACTAACCTAGCCCTACCCACCTCTATCAATTTAATTGGCGAACTATTCGTCACTATAGCATCCTTCTC
ATGATCAAACATTACAATTATCTTAATAGGCTTAAATATGCTCATCACCGCTCTCTATTCCCTCTATATATTAACTACTA
CACAACGAGGAAAACTCACATATCATTCGCACAACCTAAACCCATCCTTTACACGAGAAAACACCCTTATATCCATACAC
ATACTCCCCCTTCTCCTATTTACCTTAAACCCCAAAATTATTCTAGGACCCACGTACTGTAAATATAGTTTAAA-AAAAC
ACTAGATTGTGAATCCAGAAATAGAAGCTCAAAC-CTTCTTATTTACCGAGAAAGTAATGTATGAACTGCTAACTCTGCA
CTCCGTATATAAAAATACGGCTATCTCAACTTTTAAAGGATAGAAGTAATCCATTGGCCTTAGGAGCCAAAAA-ATTGGT
GCAACTCCAAATAAAAGTAATAAATCTATTATCCTCTTTCACCCTTGTCACACTGATTATCCTAACTTTACCTATCATTA
TAAACGTTACAAACATATACAAAAACTACCCCTATGCACCATACGTAAAATCTTCTATTGCATGTGCCTTCATCACTAGC
CTCATCCCAACTATATTATTTATCTCCTCAGGACAAGAAACAATCATTTCCAACTGACATTGAATAACAATCCAAACCCT
AAAACTATCTATTAGCTT
```

Now we can replace the sequence in the alignment with the updated string. However, we first need to convert the string back into the Biopython sequence format. For this we use the `Seq()` function:

In [16]:
```
from Bio.Seq import Seq
new_sequence = Seq(new_sequence_string)
sample_0.seq = new_sequence
```

Now let's print the alignment again and see if the sequence was changed (the first sequence should now contain N's instead of invalid characters).

In [17]:
```
print(alignment)
```

```
IUPACAmbiguousDNA() alignment with 12 rows and 898 columns
NNNNNNNATAGGAGCAACCATTCTAATAATCGCACATGGCCTTA...CTT Lemur_catta
AAGCTTCACCGGCSRYSRYSRYSRYSRYSRYCACGGGCTTA...CTT Homo_sapiens
AAGCTTCACCGGCGCAATTATCCTCATAATCGCCCACGGACTTA...CTT Pan
AAGCTTCACCGGCGCAGTTGTTCTTATAATTGCCCACGGACTTA...CTT Gorilla
AAGCTTCACCGGCGCAACCACCCTCATGATTGCCCATGGACTCA...CTT Pongo
AAGCYRYRRYGGTGCAACCGTCCTCATAATCGCCCACGGACTAA...CTT Hylobates
AAGCTTTTCCGGCGCAACCATCCTTATGATCGCTCACGGACTCA...CTT Macaca_fuscata
AAGCTTTTCTGGCGCAACCATCCTCATGATTGCTCACGGACTCA...CTT M._mulatta
AAGCTTCTCCGGCGCAACCACCCTTATAATCGCCCACGGGCTCA...CTT M._fascicularis
AAGCTYYYYYYYYYYAACTATCCTTATAGTTGCCCATGGACTCA...CTT M._sylvanus
AAGCTTCACCGGCGCAATGATCCTAATAATCGCTCACGGGTTTA...CTT Saimiri_sciureus
AAGTTTCATTGGAGCCACCACTCTTATAATTGCCCATGGCCTCA...CTT Tarsius_syrichta
```

5. Write a python function

The steps in the section above are a little lengthy and would be annoying to do separately for every single sequence in the alignment. In order to make our life easier, let's write a function that repeats these steps for every sequence in the alignment and returns the edited alignment. The basic syntax for writing a function in python is this:

In [18]:
```
def my_function(input_variable1, input_variable2):
    # do something here
    return output_variable
```

The function get's some variables as input, then computes something from them and then puts out variables. The output of a function is defined by the return statement. The names of the variables are only relevant within the function, but are not exported into the python environment.

For example if we want to write a function that calculates the sum between 2 numbers, it would look like this:

In [19]:
```
def sum_2_numbers(number1, number2):
    sum_numbers = number1 + number2
    return sum_numbers
```

You can call a function by it's name, e.g.:

In [20]:
```
sum_2_numbers(9,3)
```

Out[20]:
```
12
```

You can also parse variables as input into the function, e.g.:

In [21]:
```
first_number = 4
second_number = 13
sum_2_numbers(first_number,second_number)
```

Out[21]:
```
17
```

Now let's slightly modify the function, so that it returns both, the sum and the product of two numbers. Note that we have to list all variables we want to export after the `return` statement:

In [22]:
```
def sum_and_multiply_2_numbers(number1, number2):
    sum_numbers = number1 + number2
    product = number1 * number2
    return sum_numbers, product
```

Now let's run our new function:

In [23]:
```
sum_and_multiply_2_numbers(5,9)
```

Out[23]:
```
(14, 45)
```

Usually you want to store the output as variables. For this you need to parse the funciton output to newly defined variables. In this case, since we have two output variables we also need to define two new variables to store them as, e.g.:

In [24]:
```
output1, output2 = sum_and_multiply_2_numbers(5,9)
print(output1)
print(output2)
```

```
14
45
```

You can name the function and the input variables however you want, for example let's name the function `bla` and the input variables `X` and `Y` and define it to calculate the product of 2 numbers:

In [25]:
```
def bla(X, Y):
    product = X * Y
    return product
```

However, it makes sense to give your function sensible names, so you remember what exactly they do.

In [26]:
```
bla(9,3)
```

Out[26]:
```
27
```

You can define a function to basically do anything. Here we will define a function that incorporates the steps we did in section 4 of this tutorial. Instead of only doing these steps on a single sequence, we repeat the steps in a for-loop, iterating through all sequences, and thereby changing the whole alignment.

> **Task:** Fill in the 4 steps in the function below. You can scroll up to section 4 of the tutorial to see how each of these steps is done. Note that we don't need to define an output of our function. When changing the value of `record.seq`, it automatically get's updated in the python alignment object.

In [ ]:
```
def replace_bad_chars(alignment):
    for record in alignment:
        #1. convert sequence into string
        #2. use re.sub() function to replace invalid characters
        #3. convert the modified string back into sequence format, using the Seq(
        #4. replace the sequence in the alignment (record.seq) with the modified
```

Now apply the function to the alignment and print the alignment to see if the function worked (all `Y`, `R` and `S` should now be coded as `N`):

In [54]:
```
replace_bad_chars(alignment)
print(alignment)
```

```
IUPACAmbiguousDNA() alignment with 12 rows and 898 columns
NNNNNNNATAGGAGCAACCATTCTAATAATCGCACATGGCCTTA...CTT Lemur_catta
AAGCTTCACCGGCNNNNNNNNNNNNNNNNNNNNNNCACGGGCTTA...CTT Homo_sapiens
AAGCTTCACCGGCGCAATTATCCTCATAATCGCCCACGGACTTA...CTT Pan
AAGCTTCACCGGCGCAGTTGTTCTTATAATTGCCCACGGACTTA...CTT Gorilla
AAGCTTCACCGGCGCAACCACCCTCATGATTGCCCATGGACTCA...CTT Pongo
AAGCNNNNNNGGTGCAACCGTCCTCATAATCGCCCACGGACTAA...CTT Hylobates
AAGCTTTTCCGGCGCAACCATCCTTATGATCGCTCACGGACTCA...CTT Macaca_fuscata
AAGCTTTTCTGGCGCAACCATCCTCATGATTGCTCACGGACTCA...CTT M._mulatta
AAGCTTCTCCGGCGCAACCACCCTTATAATCGCCCACGGGCTCA...CTT M._fascicularis
AAGCTNNNNNNNNNNAACTATCCTTATAGTTGCCCATGGACTCA...CTT M._sylvanus
AAGCTTCACCGGCGCAATGATCCTAATAATCGCTCACGGGTTTA...CTT Saimiri_sciureus
AAGTTTCATTGGAGCCACCACTCTTATAATTGCCCATGGCCTCA...CTT Tarsius_syrichta
```

6. Change alignment format

Now you can output the alignment in any format you like using the `SeqIO.write()` function. Let's for example choose to export it in FASTA format:

In [27]:
```
from Bio import SeqIO
SeqIO.write(alignment, '../output_files/my_new_alignment.fasta', 'fasta')
```

Out[27]:
```
12
```

7. Write a python program

Now where we have the `replace_bad_chars()` function and know how to chnage the format of an alignment, we can build a handy little python program do do this for us for any given input alignment. We will create this program so it can be called from the command line. If you are not so interested in reformatting alignments but have a different idea of a function you would want to build into a program, I'll be happy to help you with that.

> **Task:** Find the file `alignment_formatter.py` in the data folder of this GitHub repo and open it in a text editor. Now insert the `replace_bad_chars()` function into the script. You also need to fill in a couple of other pieces of code in order to make the program work (see comments in the `alignment_formatter.py` file).

Once this is done you can call the script from your bash terminal as follows. First explore the help function:

In [56]:
```
%%bash
python alignment_formatter.py -h
```

```
usage: alignment_formatter.py [-h] --input INPUT --input_format
                              {fasta,nexus,stockholm,phylip,clustal,emboss,phyli
p-sequential,phylip-relaxed,fasta-m10,ig,maf}
                              --output OUTPUT --output_format
                              {fasta,nexus,stockholm,phylip,clustal,phylip-seque
ntial,phylip-relaxed,maf}
                              [--fix_invalid_characters]

optional arguments:
  -h, --help            show this help message and exit
  --input INPUT         Path to your input alignment file.
  --input_format {fasta,nexus,stockholm,phylip,clustal,emboss,phylip-sequential,
phylip-relaxed,fasta-m10,ig,maf}
                        Alignment format of input file.
  --output OUTPUT       Name of output alignment.
  --output_format {fasta,nexus,stockholm,phylip,clustal,phylip-sequential,phylip
-relaxed,maf}
                        Desired alignment format of output file.
  --fix_invalid_characters
                        Replace all invalid bases (not A,C,T,G,a,c,t,g,-) with
                        "N"
```

This help function output shows you all the available flags you can use with the command. The last flag `--fix_invalid_characters` is optional, which means it does not have to be provided for the program to run (if you don't provide it, the program will not replace characters in the alignment).

Now let's transform the alignment `primates_mtDNA.nex` into fasta format and replace all invalid characters with N's:

In [57]:
```
%%bash
python alignment_formatter.py --input ./primates_mtDNA.nex --input_format nexus --
```

```
New alignment written to file formatted_alignment.fasta
```