Biopython Tutorial (I): from beginner to advanced.

SATURDAY. JANUARY 09, 2021 - 13 MINS

PYTHON BIOPYTHON BIOINFORMATICS

Biopython is a Python library for reading and writing many common biological data formats and on this post we will be explaining some basic and advanced concepts and tricks. It is a extremely powerful library which provides a wide range of functions and can be used from working with sequences to evolutionary genomics or structural proteomics. And many more!

This is the first part in a series of posts in which we will be covering these contents. I hope you enjoy and learn something new today.

1. First Steps

- What is Biopython
- Installing and upgrading

2. Working with Sequences

- Creating a sequence
- Parsing a sequence file
- Counting sequence length & number of occurrences of a nucleotide
- Calculating GC-content
- Calculating molecular weight
- Get the reverse complement of a sequence, transcription & translation
- Slicing a sequence
- Concatenating sequences
- Finding the starting index of a subsequence
- Writing sequences to a file
- Converting a FASTQ file to FASTA file & other formats

3. NCBI Entrez databases

- General Guidelines
- Accessing Pubmed, Medline, Genbank & others

4. BLAST

- Running a web BLAST
- Parsing a BLAST output

5. Multiple Sequence Alignment

- Pairwise sequence alignment
- Multiple sequence alignment: creating alignment using different algorithms: ClustalW, MUSCLE…

- Reading a MSA

6. Phylogenetics

- Constructing a phylogenetic tree
- Modifying an existing tree

7. Sequence motif analysis

8. PDB: 3D structure protein analysis

- Count atoms in a PDB structure

---

1. First Steps

Our first step is installing Biopython. The easiest way is to Biopython is using `pip` package such as follows. (Please note that Biopython also requires NumPy package, which will be installed automatically if you install Biopython with `pip`).

If you already have Biopython installed you can upgrade it using `pip` aswell.

```
pip install biopython
pip install --upgrade biopython
```

You can test if Biopython is properly installed by executing:

```
import Bio
```

If no error messages appear, then everything is working and you're all set.

---

2. Working with sequences

Creating a sequence

The easiest way to create a sequence is using the Biopython `Seq` object. The bio.seq module is really useful and we will make use of it constantly. You can check the Wiki page for more information and advice. Here is a small example.

```
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
print("Sequence is", seq_example)
>>> Sequence is AGTACACTGGT
```

Parsing a sequence file

Biopython's SeqIO (Sequence Input/Output) can be used to read sequence files. The main function is `Bio.SeqIO.parse()`, which requires a filename and format and returns a SeqRecord iterator. This is an example of parsing a FASTA file.

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

`record.id` will return the identifier of the sequence. `record.seq` will return the sequence itself. `record.description` will return the sequence description.

Counting sequence length & number of occurrences of a nucleotide

Counting sequence length is really easy using `len()`. Using our previous example:

```
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
print("Sequence is", seq_example)
>>> Sequence is AGTACACTGGT
print("The length of the sequence is", len(seq_example))
>>> The length of the sequence is 11
```

You can count numer of occurrences of a certain nucleotide as follows: (Make sure to create/load sequence as we've done earlier)

```
print(seq_example.count("C"))
>>> 2
```

Calculating GC-content

GC content is a measure of the percentage of guanine (G) or cytosine (C) related to the total nitrogenous bases (A+T on DNA or A+U on RNA). GC content is important because it indicates a higher melting temperature and also key factor in bacteria taxonomy. Biopython has a `Bio.SeqUtils` package that simplifies this task. Feel free to check wiki for more information.

```
from Bio.SeqUtils import GC
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
print(GC(seq_example))
>>> 45.45
```

Calculating molecular weight

Again, molecular weight can be easily calculated using `Bio.SeqUtils` as follows:

```
from Bio.SeqUtils import molecular_weight
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
print(molecular_weight(seq_example))
>>> 3436.19
```

Get the reverse complement of a sequence, transcription & translation

Using Biopython these three processes ar really straightforward. To get the reverse complement use the single function `reverse_complement()`.

```
print("Reverse complement:", seq_example.reverse_complement())
>>> The reverse complement if the sequence is ACCAGTGTACT
```

Transcription is also a single function, `transcribe()`

```
print("Transcription:", seq_example.transcribe())
>>> Transcription: AGUACACUGGU
```

Translation is again another function, `translate()`. Please note that if your sequence is not suitable for correct translation you will recieve a warning from Biopython (our example gets this warning). This happens if your sequence is not a multiple of three, which makes partial codons on translation (we'll come back to this later). You can have more information about this biological process here

```
print("Translation:", seq_example.translate())
>>> Translation: STL
```

Slicing a sequence

With Biopython you can slice effectively sequences and extract any part of it.

```
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
print(seq_example[4:9])
>>> CACTG
```

It follows the usual indexing conventions for Python strings, with the first element of the sequence numbered 0. When you do a slice the first item is included (i.e. 4) and the last is excluded (9 in this case). We can get the third codon positions of this DNA sequence:

```
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
print(seq_example[0::3])
>>> AACG
```

Concatenating sequences

You can add any two `Seq` objects together

```
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
seq_add = Seq("TTTT")
print(seq_example + seq_add)
>>> AGTACACTGGTTTTT
```

Biopython `Seq` also has a `.join` method which can be really useful

```
from Bio.Seq import Seq
fragments = [Seq("AGTA"), Seq("CACT"), Seq("GGT")]
filler = Seq("G"*10)
print(filler.join(fragments))
>>> AGTAGGGGGGGGGGCACTGGGGGGGGGGGGT
```

Find the starting index of a subsequence

Easiest way to find starting index of a subsequence is using `find()`

```
from Bio.Seq import Seq
seq_example = Seq("AGTACACTGGT")
print("CAC index:", seq_example.find("CAC"))
>>> CAC index: 4
```

Writing sequences to a file

SeqIO (Sequence Input/Output) can also be used to write sequences to files. Our main function is `SeqIO.write()` and it allows you to save sequences as well as id or descriptions (See below for instructions on how to enter Genbank and other NCBI tools). Here is an example:

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO

rec1 = SeqRecord(
    Seq(
        "MMYQQGCFAGGTVLRLAKDLAENNRGARVLVVCSEITAVTFRGPSETHLDSMVGQALFGD"
        "GAGAVIVGSDPDLSVERPLYELVWTGATLLPDSEGAIDGHLREVGLTFHLLKDVPGLISK"
        "NIEKSLKEAFTPLGISDWNSTFWIAHPGGPAILDQVEAKLGLKEEKMRATREVLSEYGNM"
        "SSAC",
    ),
    id="gi|14150838|gb|AAK54648.1|AF376133_1",
    description="chalcone synthase [Cucumis sativus]",
)

rec2 = SeqRecord(
    Seq(
        "YPDYYFRITNREHKAELKEKFQRMCDKSMIKKRYMYLTEEILKENPSMCEYMAPSLDARQ"
        "DMVVVEIPKLGKEAAVKAIKEWGQ",
    ),
    id="gi|13919613|gb|AAK33142.1|",
    description="chalcone synthase [Fragaria vesca subsp. bracteata]",
)
```

```
rec3 = SeqRecord(
    Seq(
        "MVTVEEFRRAQCAEGPATVMAIGTATPSNCVDQSTYPDYYFRITNSEHKVELKEKFKRMC"
        "EKSMIKKRYMHLTEEILKENPNICAYMAPSLDARQDIVVVEVPKLGKEAAQKAIKEWGQP"
        "KSKITHLVFCTTSGVDMPGCDYQLTKLLGLRPSVKRFMMYQQGCFAGGTVLRMAKDLAEN"
        "NKGARVLVVCSEITAVTFRGPNDTHLDSLVGQALFGDGAAAVIIGSDPIPEVERPLFELV"
        "SAAQTLLPDSEGAIDGHLREVGLTFHLLKDVPGLISKNIEKSLVEAFQPLGISDWNSLFW"
        "IAHPGGPAILDQVELKLGLKQEKLKATRKVLSNYGNMSSACVLFILDEMRKASAKEGLGT"
        "TGEGLEWGVLFGFGPGLTVETVVLHSVAT",
    ),
    id="gi|13925890|gb|AAK49457.1|",
    description="chalcone synthase [Nicotiana tabacum]",
)

my_records = [rec1, rec2, rec3]
SeqIO.write(my_records, "seq_examples.faa", "fasta")
```

Output will be a fasta file with your sequences and records in a fasta file.

Converting a FASTQ file to FASTA file & other formats

We need to convert DNA data file formats in certain applications. You can use `SeqIO.write()` but the easiest way to perform this common task is just `SeqIO.convert()` . See this example changing GenBank format to FASTA:

```
from Bio import SeqIO
count = SeqIO.convert("ls_orchid.gbk", "genbank", "orchid_converted.fasta", "fasta"
print("Converted %i records" % count)
```

Same concept, converting from FASTQ to FASTA as follows.

```
from Bio import SeqIO
count = SeqIO.convert("ls_orchid.fastq", "fastq", "orchid_converted.fasta", "fasta"
print("Converted %i records" % count)
```

See help function to have a complete list of conversions you can do:

```
from Bio import SeqIO
help(SeqIO.convert)
```

---

3. NCBI Entrez databases

General Guidelines

Entrez is a data retrieval system that provides users access to NCBI’s databases such as PubMed, GenBank, GEO, and many others. Using Biopython’s module `Bio.Entrez` allows you to access Entrez and search & download records from within a Python script. You must provide an email to access Entrez - NCBI can block anonymous requests. Also, `Bio.Entrez` might not be the best option to download huge amounts of data as it can clog the servers.

```
from Bio import Entrez
Entrez.email = "davidboo@example.com"
```

Accessing PubMed, Medline, Genbank & others

To search any of these databases, we use `Bio.Entrez.esearch()` module, which requires some parameters:

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

Where db are databases such as Pubmed, nucleotide, protein, snp, omim, unigene… keywords are what you are interested in looking at (organisms, Human, Biopython, urea…) and retmax are the number of identifies returned.

Let’s do an example accessing PubMed. We will be searching in PubMed for 20 publications that include breast cancer in their title.

```
from Bio import Entrez
Entrez.email = "davidboo@example.com"
handle = Entrez.esearch(db="pubmed", term="breast[title] AND cancer[title]", retmax
records = Entrez.read(handle)
print(records["IdList"])
```

If you want to retrieve a full record from Entrez, then you should use `Bio.Entrez.efetch()`

```
from Bio import Entrez
Entrez.email = "davidboo@example.com"
handle = Entrez.efetch(db="nucleotide", id="KY398842.1", rettype="gb", retmode="tex
print(handle.read())
```

If you fetch the record in one of the formats accepted by `Bio.SeqIO`, you could directly parse it into a `SeqRecord`:

```
from Bio import SeqIO
from Bio import Entrez
Entrez.email = "davidboo@example.com"
handle = Entrez.efetch(db="nucleotide", id="KY398842.1", rettype="gb", retmode="tex
record = SeqIO.read(handle, "genbank")
handle.close()
print(record.id)
>>> KY398842.1
print(record.name)
>>> KY398842
print(record.description)
```

```
>>> Rabbit hemorrhagic disease virus isolate RHDV BZ/2015 capsid structural protein
record.seq
>>> Seq('ATGGAGGGCAAAGCCCGCACAGCGCCGCAAGGCGAAGCAGCAGGCACTGCTAC...TGA')
```

That's all for today! I hope you got an idea on how to use basic Biopython for sequence works.

Thank you for reading. Feel free to check out our next post for more information on Biopython and many other things.

Let me know if you have any feedback or issues, I'd love to hear from you!

Related Posts

- Rosalind: Python solutions to common problems in Bioinformatics
- Biopython Tutorial (II): from beginner to advanced.
- Project Euler: a Python (beginner friendly) approach to complex mathematical challenges

David Boo
Where biology, informatics & statistics intersect

Tweet Share