

# Biopython

## What is biopython?

Biopython is a collection of python modules that contain code for manipulating biological data. Many handle sequence data and common analysis and processing of the data including reading and writing all common file formats. Biopython will also run blast for you and parse the output into objects inside your script. This requires just a few lines of code.

## Installing Biopython

This is very straightforward once you have anaconda or miniconda installed. I use miniconda because it's smaller. We are going to use `sudo`, because this will give us permission to install in the 'correct' directory python is expecting to find the modules. Other users will be able to use it too. Using `sudo` can cause problems, but it's ok here. You will need the administrator password for the machine. If you don't have this, ask the person who does administration on your machine.

```
% conda install biopython
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

```
## Package Plan ##
```

```
environment location: /Users/smr/opt/anaconda3
```

```
added / updated specs:
```

```
- biopython
```

The following packages will be downloaded:

package	build	
biopython-1.78	py39h9ed2024_0	2.1 MB
conda-22.9.0	py39hecd8cb5_0	884 KB
Total:		3.0 MB

The following NEW packages will be INSTALLED:

```
biopython          pkgs/main/osx-64::biopython-1.78-py39h9ed2024_0
```

The following packages will be UPDATED:

```
conda 4.14.0-py39hecd8cb5_0 --> 22.9.0-py39hecd8cb5_0
```

```
Proceed ([y]/n)?
```

```
Downloading and Extracting Packages
```

```
biopython-1.78 | 2.1 MB |
```

```
#####
```

```
| 100%
```

```
conda-22.9.0 | 884 KB |
```

```
#####
```

```
| 100%
```

```
Preparing transaction: done
```

```
Verifying transaction: done
```

```
Executing transaction: done
```

```
Retrieving notices: ...working... done
```

See if the install worked

```
python3
>>> import Bio
>>> print(Bio.__version__)
1.78
```

If we get no errors, biopython is installed correctly.

## Biopython documentation

[Biopython wiki page](#)

[Getting started](#)

[Biopython tutorial](#)

[Complete tree of Biopython Classes](#)

## Working with DNA and protein sequences

This is the core of biopython. And uses the Seq object. Seq is part of Bio. This is denoted Bio.Seq

```
#!/usr/bin/env python3
import Bio.Seq
seqobj = Bio.Seq.Seq('ATGCGATCGAGC')
print(f"{seqobj} has {len(seqobj)} nucleotides")
```

Note: Sometimes you might have to convert an object to string to get sequence `seq_str = str(seqobj)`. The Seq Object predicts that if a user writes `print(seqobj)` they will want to print the sequence string not the entire Seq Object. Likewise, the Seq Object predicts that if a user writes `len(seqobj)` they will want to calculate the length of the sequence not the length of the entire Seq Object

produces

```
ATGCGATCGAGC has 12 nucleotides
```

## From ... import ...

Another way to import modules is with `from ... import ...`. This saves typing the Class name every time. Bio.Seq is the class name. Bio is the superclass. Seq is a subclass inside Bio. It's written Bio.Seq. Seq has several different subclasses, of which one is called Seq. So we have Bio.Seq.Seq. To make the creation simpler, we call Seq() after we import with `from ... import ...` like this

```
#!/usr/bin/env python3
from Bio.Seq import Seq
seqobj=Seq('ATGCGATCGAGC')
protein = seqobj.translate()
print(f'{seqobj} translates to {protein}')
```

produces

```
ATGCGATCGAGC translates to MRSS
```

## Extracting a subsequence

You can use a range [0:3] to get the first codon

Visit [biopython.org](http://biopython.org) to read about [Slicing a sequence](#)

```
>>> seqobj=Seq('ATGCGATCGAGC')
>>> seqobj[0:3]
Seq('ATG')
>>> print(seqobj[0:3])
ATG
```

Let's use Regular expressions in conjunction with BioPython to get every codon

```
>>> seqobj=Seq( 'ATGCGATCGAGC' )
>>> import re
>>> for codon in re.findall(r"(.{3})",str(seqobj)):
...     print(codon)
...
ATG
CGA
TCG
AGC
>>>
```

The Seq Object has not predicted that if we use seqobj as input to `findall()` that we want to search just the sequence. But it has predicted that if we use the `str()` we want to return the sequence that is contained within our object.

## Data types

The Seq Object predicts that we want a string when we `print()` our seqobj or if we try to calculate `len()` or if we try to take a substr `seqobj[0:3]` of our seqobj. The authors have coded this functionality into the Class rules. They did not predict, or write into the Class rules that if we use `findall()` that we want to search just the sequence. The Class does not know how to handle this. But it has predicted that if we use the `str()` we want to return the sequence that is contained within our object.

```
>>> seqobj=Seq( 'ATGCGATCGAGC' )
>>> type(seqobj)
<class 'Bio.Seq.Seq'>
>>> str(seqobj)
'ATGCGATCGAGC'
>>> type(str(seqobj))
<class 'str'>
```

## Read a FASTA file

Earlier in the course we were learning how to read a fasta file line by line. We are going to go over the BioPython way to do this. `SeqIO.parse()` is the main method for reading from almost any file format. The examples will use [seq.nt.fa](#):

```

>seq1
AAGAGCAGCTCGCGCTAATGTGATAGATGGCGGTAAAGTAAATGTCCTATGGGCCACCAATTATGGTGTATGAGTGAATCTCTGGTC
CGAGATTCA
CTGAGTAACTGCTGTACACAGTAGTAACACGTGGAGATCCCATAAGCTTCACGTGTGGTCCAATAAAACACTCCGTTGGTCAAC

>seq2
GCCACAGAGCCTAGGACCCCAACCTAACCTAACCTAACCTAACCTACAGTTTGATCTTAACCATGAGGCTGAGAAGCGATGTCCTGA
CCGGCCTGT
CCTAACCGCCCTGACCTAACCGGCTTGACCTAACCGCCCTGACCTAACCGGCTAACCTAACCAAACCGTGAAAAAAGGAATCT

>seq3
ATGAAAGTTACATAAAGACTATTCGATGCATAAATAGTTCAGTTTGGAAACTTACATTTTGTAAAGTCAGGTACTTGTGTATAAT
ATCAACTAA
AT

>seq4
ATGCTAACCAAAGTTTCAGTTCGGACGTGTCGATGAGCGACGCTCAAAAAGGAAACAACATGCCAAATAGAAACGATCAATTCGGCG
ATGGAAATC
AGAACAACGATCAGTTTGGAAATCAAAATAGAAATAACGGGAACGATCAGTTTAATAACATGATGCAGAATAAAGGGAATAATCAAT
TTAATCCAG
GTAATCAGAACAGAGGT

```

Get help on the parse() method with

```

>>> from Bio import SeqIO
>>> help(SeqIO.parse)

Help on function parse in module Bio.SeqIO:

parse(handle, format, alphabet=None)
    Turns a sequence file into an iterator returning SeqRecords.

    - handle    - handle to the file, or the filename as a string
                  (note older versions of Biopython only took a handle).
    - format    - lower case string describing the file format.
    - alphabet  - optional Alphabet object, useful when the sequence type
                  cannot be automatically inferred from the file itself
                  (e.g. format="fasta" or "tab")

...

```

Here's a script to read fasta records and print out some information

```
#!/usr/bin/env python3
from Bio import SeqIO
for seq_record in SeqIO.parse("../files/seq.nt.fa", "fasta"): # give filename and
    format
    print('ID',seq_record.id)
    print('Sequence',seq_record.seq)
    print('Length',len(seq_record))
```

Prints this output

```
ID seq1
Sequence
AAGAGCAGCTCGCGCTAATGTGATAGATGGCGGTAAAGTAAATGTCTATGGGCCACCAATTATGGTGTATGAGTGAATCTCTGGTC
CGAGATTCACTGAGTAACTGCTGTACACAGTAGTAACACGTGGAGATCCCATAGCTTCACGTGTGGTCCAATAAAACACTCCGTTG
GTCAAC
Length 180
ID seq2
Sequence
GCCACAGAGCCTAGGACCCCAACCTAACCTAACCTAACCTACAGTTTGATCTTAACCATGAGGCTGAGAAGCGATGTCTCTGA
CCGGCCTGTCTAACC GCCCTGACCTAACCGGCTTGACCTAACCGCCCTGACCTAACCGGCTAACCTAACCAAACCGTGAAAAAG
GAATCT
Length 180
ID seq3
Sequence
ATGAAAGTTACATAAAGACTATTTCGATGCATAAATAGTTCAGTTTGTGAAAACCTACATTTTGTAAAGTCAGGTACTTGTGTATAAT
ATCAACTAAAT
Length 98
ID seq4
Sequence
ATGCTAACCAAAGTTTCAGTTCGGACGTGTGATGAGCGACGCTCAAAAAGGAAACAACATGCCAAATAGAAACGATCAATTCGGCG
ATGGAAATCAGAACAACGATCAGTTTGGAAATCAAAATAGAAATAACGGGAACGATCAGTTTAATAACATGATGCAGAATAAAGGGA
ATAATCAATTTAATCCAGGTAATCAGAACAGAGGT
Length 209
```

## How do you know what methods and attributes are available?

In the last example we used the `id()` and `seq()`. How do we find out that we could use these or what are other options are?

You can use option+tab in the interpreter to find out. Type the object then a '.' then option+tab. You will get a list of attributes and methods you can use with this specific object.

```
>>> from Bio import SeqIO
>>> for seq_record in SeqIO.parse("../files/seq.nt.fa", "fasta"):
...     print(seq_record.
seq_record.annotations          seq_record.id          seq_record.seq
seq_record.dbxrefs              seq_record.letter_annotations  seq_record.translate(
seq_record.description          seq_record.lower(          seq_record.upper(
seq_record.features             seq_record.name
seq_record.format(              seq_record.reverse_complement(
...     print(seq_record.
```

## Seq Object vs SeqRecord Object

The Seq Object and the SeqRecord Object two Objects are not the same. As you have seen we can directly print the sequence that is stored within a `Seq` Object. But this is not possible with `SeqRecord`. You need to use the `seq()` method to retrieve just the sequence bit of the `SeqRecord` Object.

```
>>> from Bio.Seq import Seq
>>> seqobj=Seq('ATGCGATCGAGC')
>>> print(seqobj)
ATGCGATCGAGC
>>>
>>> type(seqobj)
<class 'Bio.Seq.Seq'>
>>>
>>> from Bio import SeqIO
>>> filename = "../files/seq.nt.fa"
>>> for seq_record in SeqIO.parse(filename, "fasta"):
...     type(seq_record)
...     print(seq_record.seq)
...     print(seq_record)
...
<class 'Bio.SeqRecord.SeqRecord'>
AAGAGCAGCTCGCGCTAATGTGATAGATGGCGGTAAAGTAAATGTCCTATGGGCCACCAATTATGGTGTATGAGTGAATCTCTGGTC
CGAGATTCACTGAGTAACTGCTGTACACAGTAGTAACACGTGGAGATCCCATAGCTTCACGTGTGGTCCAATAAAACACTCCGTTG
GTCAAC
ID: seq1
Name: seq1
Description: seq1
Number of features: 0
Seq('AAGAGCAGCTCGCGCTAATGTGATAGATGGCGGTAAAGTAAATGTCCTATGGGC...AAC')
<class 'Bio.SeqRecord.SeqRecord'>
```

```
GCCACAGAGCCTAGGACCCCAACCTAACCTAACCTAACCTAACCTACAGTTTGATCTTAACCATGAGGCTGAGAAGCGATGTCCTGA
CCGGCCTGTCTTAACCGCCCTGACCTAACCGGCTTGACCTAACCGCCCTGACCTAACCGGCTAACCTAACCAAACCGTGAAAAAAG
GAATCT
# ... etc
```

Here is another example of opening a FASTA file, retrieving each sequence record, and doing something the data. We are going to translate each sequence record

```
#!/usr/bin/env python3
from Bio import SeqIO
filename = "../files/seq.nt.fa"
for seq_record in SeqIO.parse(filename, "fasta"):
    print('ID', seq_record.id)
    print(f'len {len(seq_record)}')
    print(f'translation {seq_record.seq.translate(to_stop=False)}')
```

We added the translation of the DNA sequence into protein  
Output:

```
ID seq1
len 180
translation KSSSR*CDRWR*SKCPMGHQLWCMSESLVRDSLNCCTQ**HVEIP*ASRVVQ*NTPLVN
ID seq2
len 180
translation ATEPRTP*T*PNLT*PTV*S*P*G*EAMS*PACPNRPDLTGLT*PP*PNQANLTKP*KKES
ID seq3
len 98
translation MKVT*RLFDA*IVQF*KLTFC*SQVLVYNIN*
ID seq4
len 209
translation MLTKVSVRTCR*ATLKKETTTCQIETINSAMEIRTTISLEIKIEITGTISLIT*CRIKGIINLIQVIRTE
```

Because one of our sample sequences is not a complete CDS we will get this message from biopython

```
/Users/smr/opt/anaconda3/lib/python3.9/site-packages/Bio/Seq.py:2334: BiopythonWarning:
Partial codon, len(sequence) not a multiple of three. Explicitly trim the sequence or
add trailing N before translation. This may become an error in future.
warnings.warn(
```

This is displayed to standard error and standard out, and therefore will not affect the contents if redirected from standard out into a file.

```
% python3 biopython_translate.py > tmp
```



```
/Users/smr/opt/anaconda3/lib/python3.9/site-packages/Bio/Seq.py:2334: BiopythonWarning:
Partial codon, len(sequence) not a multiple of three. Explicitly trim the sequence or
add trailing N before translation. This may become an error in future.
```

```
warnings.warn(
% cat tmp
ID seq1
len 180
translation KSSSR*CDRWR*SKCPMGHQLWCMSESLVRDSLSNCCTQ**HVEIP*ASRVVQ*NTPLVN
ID seq2
len 180
translation ATEPRTP*T*PNLT*PTV*S*P*G*EAMS*PACPNRPDLTGLT*PP*PNQANLTKP*KKES
ID seq3
len 98
translation MKVT*RLFDA*IVQF*KLTF*SQVLVYNIN*
ID seq4
len 209
translation MLTKVSVR*TCR*ATLKKETT*CIETINSAMEIRTTISLEIKIEITGTISLIT*CRKIGIINLIQVIRTE
```

## Convert FASTA file to Python dictionary in one line

`Bio.SeqIO.to_dict()` reads the entire FASTA file into memory and stores the contents in a dictionary.

```
>>> from Bio import SeqIO
>>> id_dict = SeqIO.to_dict(SeqIO.parse('../files/seq.nt.fa', 'fasta'))
>>> id_dict
{'seq1':
SeqRecord(seq=Seq('AAGAGCAGCTCGCGCTAATGTGATAGATGGCGGTAAAGTAAATGTCCTATGGGC...AAC'),
id='seq1', name='seq1', description='seq1', dbxrefs=[]), 'seq2':
SeqRecord(seq=Seq('GCCACAGAGCCTAGGACCCCAACCTAACCTAACCTAACCTACAGTTTGA...TCT'),
id='seq2', name='seq2', description='seq2', dbxrefs=[]), 'seq3':
SeqRecord(seq=Seq('ATGAAAGTTACATAAAGACTATTCGATGCATAAATAGTTTCAGTTTGGAAACTT...AAT'),
id='seq3', name='seq3', description='seq3', dbxrefs=[]), 'seq4':
SeqRecord(seq=Seq('ATGCTAACCAAAGTTTCAGTTCGACGTGTCGATGAGCGACGCTCAAAAAGGAA...GGT'),
id='seq4', name='seq4', description='seq4', dbxrefs=[])}
```

Let's retrieve some info from our new dictionary

```
>>> id_dict['seq4']
SeqRecord(seq=Seq('ATGCTAACCAAAGTTTCAGTTCGGACGTGTCGATGAGCGACGCTCAAAAAGGAA...GGT'),
id='seq4', name='seq4', description='seq4', dbxrefs=[])
>>> id_dict['seq4'].seq
Seq('ATGCTAACCAAAGTTTCAGTTCGGACGTGTCGATGAGCGACGCTCAAAAAGGAA...GGT')
>>> str(id_dict['seq4'].seq)
'ATGCTAACCAAAGTTTCAGTTCGGACGTGTCGATGAGCGACGCTCAAAAAGGAAACAACATGCCAAATAGAAACGATCAATTCGGC
GATGGAAATCAGAACACGATCAGTTTGGAAATCAAAATAGAAATAACGGGAACGATCAGTTTAATAACATGATGCAGAATAAAGGG
AATAATCAATTTAATCCAGGTAATCAGAACAGAGGT'
```

need to use this format to get the string of the sequence: `str(id_dict['seq4'].seq)`

## Seq methods

Visit [biopython.org](http://biopython.org) to read how [Sequences act like strings](#)

```
from Bio.Seq import Seq
seqobj.count("A") # counts how many As are in sequence
seqobj.find("ATG") # find coordinate of ATG (-1 for not found)
```

OR, as mentioned earlier in the interpreter you can use option+tab to find out what methods are available:

```
>>> from Bio.Seq import Seq
>>> seqobj=Seq('ATGCGATCGAGC')
>>> seqobj.
seqobj.alphabet          seqobj.find(           seqobj.rstrip(
seqobj.transcribe(      seqobj.lower(         seqobj.split(
seqobj.back_transcribe( seqobj.lstrip(        seqobj.startswith(
seqobj.translate(       seqobj.lstrip(        seqobj.strip(
seqobj.complement(      seqobj.reverse_complement(
seqobj.ungap(           seqobj.strip(
seqobj.count(           seqobj.rfind(         seqobj.tomutable(
seqobj.upper(           seqobj.rsplit(        seqobj.tostring(
seqobj.count_overlap(   seqobj.rstrip(
seqobj.endswith(
>>> seqobj.
```

AND, you can use the `help()` in the interpreter to find out more:

```
>>> help(seqobj.count_overlap)
```

Help on method count\_overlap in module Bio.Seq:

count\_overlap(sub, start=0, end=9223372036854775807) method of Bio.Seq.Seq instance  
Return an overlapping count.

For a non-overlapping search use the count() method.

Returns an integer, the number of occurrences of substring  
argument sub in the (sub)sequence given by [start:end].  
Optional arguments start and end are interpreted as in slice  
notation.

Arguments:

- sub - a string or another Seq object to look for
- start - optional integer, slice start
- end - optional integer, slice end

e.g.

```
>>> from Bio.Seq import Seq
>>> print(Seq("AAAA").count_overlap("AA"))
3
>>> print(Seq("ATATATATA").count_overlap("ATA"))
4
>>> print(Seq("ATATATATA").count_overlap("ATA", 3, -1))
1
```

Where substrings do not overlap, should behave the same as  
the count() method:

:

## SeqRecord objects

SeqIO.Parse generates Bio.SeqRecord.SeqRecord objects. These are annotated Bio.Seq.Seq objects.

Main attributes:

- id - Identifier such as a locus tag (string)
- seq - The sequence itself (Seq object or similar)

Access these with `sr.id` and `sr.seq`. `str(sr.seq)` gets the actual sequence string.

Additional attributes:

- name - Sequence name, e.g. gene name (string)

- description - Additional text (string)
- dbxrefs - List of database cross references (list of strings)
- features - Any (sub)features defined (list of SeqFeature objects)
- annotations - Further information about the whole sequence (dictionary). Most entries are strings, or lists of strings.
- letter\_annotations - Per letter/symbol annotation (restricted dictionary). This holds Python sequences (lists, strings or tuples) whose length matches that of the sequence. A typical use would be to hold a list of integers representing sequencing quality scores, or a string representing the secondary structure.

SeqRecord objects have .format() to convert to a string in various formats

```
>>> for seq_record in SeqIO.parse("../files/seq.nt.fa", "fasta"):
...     seq_record.format('fasta')
...
'>seq1\nAAGAGCAGCTCGCGCTAATGTGATAGATGGCGGTAAAGTAAATGTCCTATGGGCCACCAA\nTTATGGTGTATGAGTGA
ATCTCTGGTCCGAGATTCAGTAACTGCTGTACACAGTAG\nTAACACGTGGAGATCCCATAAGCTTCACGTGTGGTCCAATAA
AACACTCCGTTGGTCAAC\n'
```

In the interpreter:

```
...     seq_record.
seq_record.annotations      seq_record.id      seq_record.seq
seq_record.dbxrefs         seq_record.letter_annotations  seq_record.translate(
seq_record.description     seq_record.lower(   seq_record.upper(
seq_record.features        seq_record.name
seq_record.format(         seq_record.reverse_complement(
```

## Retrieving annotations from GenBank file

To read sequences from a genbank file instead, not much changes.

```
#!/usr/bin/env python3
from Bio import SeqIO
for seq_record in SeqIO.parse("../files/sequence.gb", "genbank"):
    print('ID', seq_record.id)
    print('Sequence', str(seq_record.seq)[0:60], '...')
    print('Length', len(seq_record))
```

Output:

ID NM\_204156.1

Sequence GGCCCCGGCCGGTGGGGCGGGTTGCGTTGCGCTGCGCGGCGGTAGGGTCTGCGGCCGTGG ...

Length 3193

## File Format Conversions

Many are straightforward, others are a little more complicated because the alphabet can't be determined from the data. It's usually easier to go from richer formats to simpler ones.

```
#!/usr/bin/env python3
from Bio import SeqIO
fasta_records = SeqIO.parse("../files/seq.nt.fa", "fasta")
count = SeqIO.write(fasta_records, '../files/seqs.tab', 'tab')
```

Produces

```
% more seqs.tab
seq1
AAGAGCAGCTCGCGCTAATGTGATAGATGGCGGTAAAGTAAATGTCCTATGGGCCACCAATTATGGTGTATGAGTGAATCTCTGGTC
CGAGATTCACTGAGTAACTGCTGTACACAGTAGTAACACGTGGAGATCCCATAGCTTCACGTGTGGTCCAATAAAACACTCCGTTG
GTCAAC
seq2
GCCACAGAGCCTAGGACCCCAACCTAACCTAACCTAACCTAACCTACAGTTTGATCTTAACCATGAGGCTGAGAAGCGATGTCCTGA
CCGGCCTGTCTTAACCGCCCTGACCTAACCGGCTTGACCTAACCGCCCTGACCTAACCGGCTAACCTAACCAAAACCGTGAAAAAG
GAATCT
seq3
ATGAAAGTTACATAAAGACTATTCGATGCATAAATAGTTCAGTTTGTGAAAACCTACATTTTGTAAAGTCAGGTACTTGTGTATAAT
ATCAACTAAAT
seq4
ATGCTAACCAAAAGTTTCAGTTCGGACGTGTCGATGAGCGACGCTCAAAAAGGAAACAACATGCCAAATAGAAACGATCAATTCGGCG
ATGGAAATCAGAACAACGATCAGTTTGAAATCAAAATAGAAATAACGGGAACGATCAGTTTAATAACATGATGCAGAATAAAGGGA
ATAATCAATTTAATCCAGGTAATCAGAACAGAGGT
```

Even easier is the `convert()` method. Let's try FASTQ to FASTA.

```
#!/usr/bin/env python3
from Bio import SeqIO
count = SeqIO.convert('../files/pfb.fastq', 'fastq', '../files/pfb.converted.fa',
                       'fasta')
```

Was that easy or what?!?!?!?

# Parsing BLAST output

For simple parsing, or non BioPython parsing of NCBI BLAST results, use output formatted in tab-separated columns (`-outfmt 6` or `-outfmt 7`) Both these formats are customizable when running the BLAST locally.

If you want to parse the full output of BLAST with biopython, it's necessary work with **XML** formatted BLAST output `-outfmt 5`.

You can get biopython to run the blast for you too. See `Bio.NCBIWWW`

To parse the output, you'll write something like this

```
#!/usr/bin/env python3
from Bio.Blast import NCBIXML
result_handle = open("../files/UTKBKAM5014-Alignment.xml")
blast_records = NCBIXML.parse(result_handle)
for blast_record in blast_records:
    query_id = blast_record.query_id
    for alignment in blast_record.alignments:
        for hsp in alignment.hsps:
            if hsp.expect < 1e-10:
                print(f'qid: {query_id} hit_id: {alignment.title} E: {hsp.expect}' )
                # print(query_id, alignment.title, hsp.expect, sep="\t" ) # print tab
delimited results table
```

Output:

```
qid: Query_26141 hit_id: sp|Q13547.1| RecName: Full=Histone deacetylase 1; Short=HD1
[Homo sapiens] >sp|Q5RAG0.1| RecName: Full=Histone deacetylase 1; Short=HD1 [Pongo
abelii] E: 0.0
qid: Query_26141 hit_id: sp|O09106.1| RecName: Full=Histone deacetylase 1; Short=HD1
[Mus musculus] E: 0.0
qid: Query_26141 hit_id: sp|Q4QQW4.1| RecName: Full=Histone deacetylase 1; Short=HD1
[Rattus norvegicus] E: 0.0
qid: Query_26141 hit_id: sp|Q32PJ8.1| RecName: Full=Histone deacetylase 1; Short=HD1
[Bos taurus] E: 0.0
qid: Query_26141 hit_id: sp|P56517.1| RecName: Full=Histone deacetylase 1; Short=HD1
[Gallus gallus] E: 0.0
qid: Query_26141 hit_id: sp|O42227.1| RecName: Full=Probable histone deacetylase 1-B;
Short=HD1-B; AltName: Full=RPD3 homolog [Xenopus laevis] E: 0.0
... etc
```

About BLAST Search Report and BioPython:

- `blast_records` (type `<class 'generator'>`) can contain handle multiple queries (the sequence you are using as input)
- The results for each query are considered a `blast_record` ()
- Each `blast_record` will have info about the query, like `blast_record.query_id`
- Each `blast_record` will have information about each hit.
- A Hit is considered an `alignment` (`<class 'Bio.Blast.Record.Alignment'>`)
- An `alignment` has the following info: `alignment.accession`, `alignment.hit_id`, `alignment.length`, `alignment.hit_def`, `alignment.hsps`, `alignment.title`
- Each `alignment` will have 1 or more `hsp` (`<class 'Bio.Blast.Record.HSP'>`).
- An HSP is a "high scoring pair" or a series of smaller alignments that make up the complete alignment.
- `hsp` have the following info: `hsp.align_length`, `hsp.frame`, `hsp.match`, `hsp.query`, `hsp.sbjct`, `hsp.score`, `hsp.bits`, `hsp.gaps`, `hsp.num_alignments`, `hsp.query_end`, `hsp.sbjct_end`, `hsp.strand`, `hsp.expect`, `hsp.identities`, `hsp.positives`, `hsp.query_start`, `hsp.sbjct_start`

[Download](#) [GenPept](#) [Graphics](#)
**RecName: Full=Histone deacetylase 1; Short=HD1 [Homo sapiens]**Sequence ID: [Q13547.1](#) Length: **482** Number of Matches: **1**[See 1 more title\(s\)](#)Range 1: 1 to 482 [GenPept](#) [Graphics](#)[Next Match](#) [Previous Match](#)

Score	Expect	Method	Identities	Positives	Gaps
1008 bits(2607)	0.0	Compositional matrix adjust.	482/482(100%)	482/482(100%)	0/482(0%)
Query 1	MAQTQGTTRRKVCYYYDGDVGNYYYGQGHMPKPHRIRMTHNLLLNYGLYRKMEIYRPHKAN				60
Sbjct 1	MAQTQGTTRRKVCYYYDGDVGNYYYGQGHMPKPHRIRMTHNLLLNYGLYRKMEIYRPHKAN				60
Query 61	AEEMTKYHSDDYIKFLRSIRPDNMSEYSKQMQRFNVEDCPVFDGLFEFCQLSTGGSVAS				120
Sbjct 61	AEEMTKYHSDDYIKFLRSIRPDNMSEYSKQMQRFNVEDCPVFDGLFEFCQLSTGGSVAS				120
Query 121	AVKLNKQQTDAIVNWAGGLHHAKKSEASGFCYVNDIVLAILELLKYHQRVLYIDIDIHHG				180
Sbjct 121	AVKLNKQQTDAIVNWAGGLHHAKKSEASGFCYVNDIVLAILELLKYHQRVLYIDIDIHHG				180
Query 181	DGVEEAFYTTDRVMTVSFHKYGEYFPGTGDLRDIGAGKGKYYAVNYPLRDGIDDESIEAI				240
Sbjct 181	DGVEEAFYTTDRVMTVSFHKYGEYFPGTGDLRDIGAGKGKYYAVNYPLRDGIDDESIEAI				240
Query 241	FKPVMSKVMEMFQPSAVVLQCGSDSLSGDRLGCFNLTIKGHAKCVEFVKSFNLPMLMLGG				300
Sbjct 241	FKPVMSKVMEMFQPSAVVLQCGSDSLSGDRLGCFNLTIKGHAKCVEFVKSFNLPMLMLGG				300
Query 301	GGYTIRNVARCWTYETAVALDTEIPNELPYNDYFEYFGPDFKLHISPSNMTNQNTNEYLE				360
Sbjct 301	GGYTIRNVARCWTYETAVALDTEIPNELPYNDYFEYFGPDFKLHISPSNMTNQNTNEYLE				360
Query 361	KIKQRLFENLRMLPHAPGVQMQAIPEDAIPESGDEDEDDPKRISICSSDKRIACEEEF				420
Sbjct 361	KIKQRLFENLRMLPHAPGVQMQAIPEDAIPESGDEDEDDPKRISICSSDKRIACEEEF				420
Query 421	SDSEEEGEGGRKNSSNFKKAKRVKTEDEKEKDPEEKKEVTEEEKTKEEKPEAKGVKEEVK				480
Sbjct 421	SDSEEEGEGGRKNSSNFKKAKRVKTEDEKEKDPEEKKEVTEEEKTKEEKPEAKGVKEEVK				480
Query 481	LA 482				
Sbjct 481	LA 482				

Sample of BLAST XML output:

```

<Iteration>
  <Iteration_iter-num>1</Iteration_iter-num>
  <Iteration_query-ID>Query_26141</Iteration_query-ID>
  <Iteration_query-def>CAG46518.1 HDAC1 [Homo sapiens]</Iteration_query-def>
  <Iteration_query-len>482</Iteration_query-len>
<Iteration_hits>
<Hit>
  <Hit_num>1</Hit_num>
  <Hit_id>sp|Q13547.1|</Hit_id>

```



```

    <Hit_def>RecName: Full=Histone deacetylase 1; Short=HD1 [Homo sapiens]
    &gt;sp|Q5RAG0.1| RecName: Full=Histone deacetylase 1; Short=HD1 [Pongo abelii]
  </Hit_def>
    <Hit_accession>Q13547</Hit_accession>
    <Hit_len>482</Hit_len>
    <Hit_hsps>
      <Hsp>
        <Hsp_num>1</Hsp_num>
        <Hsp_bit-score>1008.82</Hsp_bit-score>
        <Hsp_score>2607</Hsp_score>
        <Hsp_evalue>0</Hsp_evalue>
        <Hsp_query-from>1</Hsp_query-from>
        <Hsp_query-to>482</Hsp_query-to>
        <Hsp_hit-from>1</Hsp_hit-from>
        <Hsp_hit-to>482</Hsp_hit-to>
        <Hsp_query-frame>0</Hsp_query-frame>
        <Hsp_hit-frame>0</Hsp_hit-frame>
        <Hsp_identity>482</Hsp_identity>
        <Hsp_positive>482</Hsp_positive>
        <Hsp_gaps>0</Hsp_gaps>
        <Hsp_align-len>482</Hsp_align-len>

        <Hsp_qseq>MAQTQGTRRKVCYYYDGDVGNYYYGQGHMPKPHRIRMTHNLLLNYGLYRKMEIYRPHKANAEEMTKYHSDDYIKFL
        RSIRPDNMSEYSKQMQRFNVGEDCPVFDGLFEFCQLSTGGSVASAVKLNKQQTDI AVNWAGGLHHAKKSEASGFCYVNDIVLAILEL
        LKYHQRVLYIDIDIHHGDGVVEAFYTTDRVMTVSFHKYGEYFPGTGLRDI GAGKGKYYAVNYPLRDGIDDESIEAIFKPVMSKVME
        MFQPSAVVLQCGSDSLSGDRLGCFNLTIKGHAKCVEFVKSFNLPMLMLGGGGYTIRNVARCWYETAVALDTEIPNELPYNDYFEYF
        GPDFKLHISPSNMTNQNTNEYLEKIKQRLFENLRMLPHAPGVQMQAIPEDAIPESGDEDEDDPKRISICSSDKRIACEEEFS DSE
        EEGEGGRKNSSNFKKAKRVKTEDEKEKDPEEKKEVTEEEKTKEEKPEAKGVKEEVKLA</Hsp_qseq>

        <Hsp_hseq>MAQTQGTRRKVCYYYDGDVGNYYYGQGHMPKPHRIRMTHNLLLNYGLYRKMEIYRPHKANAEEMTKYHSDDYIKFL
        RSIRPDNMSEYSKQMQRFNVGEDCPVFDGLFEFCQLSTGGSVASAVKLNKQQTDI AVNWAGGLHHAKKSEASGFCYVNDIVLAILEL
        LKYHQRVLYIDIDIHHGDGVVEAFYTTDRVMTVSFHKYGEYFPGTGLRDI GAGKGKYYAVNYPLRDGIDDESIEAIFKPVMSKVME
        MFQPSAVVLQCGSDSLSGDRLGCFNLTIKGHAKCVEFVKSFNLPMLMLGGGGYTIRNVARCWYETAVALDTEIPNELPYNDYFEYF
        GPDFKLHISPSNMTNQNTNEYLEKIKQRLFENLRMLPHAPGVQMQAIPEDAIPESGDEDEDDPKRISICSSDKRIACEEEFS DSE
        EEGEGGRKNSSNFKKAKRVKTEDEKEKDPEEKKEVTEEEKTKEEKPEAKGVKEEVKLA</Hsp_hseq>

        <Hsp_midline>MAQTQGTRRKVCYYYDGDVGNYYYGQGHMPKPHRIRMTHNLLLNYGLYRKMEIYRPHKANAEEMTKYHSDDYI
        KFLRSIRPDNMSEYSKQMQRFNVGEDCPVFDGLFEFCQLSTGGSVASAVKLNKQQTDI AVNWAGGLHHAKKSEASGFCYVNDIVLAI
        LELLKYHQRVLYIDIDIHHGDGVVEAFYTTDRVMTVSFHKYGEYFPGTGLRDI GAGKGKYYAVNYPLRDGIDDESIEAIFKPVMSK
        VMEMFQPSAVVLQCGSDSLSGDRLGCFNLTIKGHAKCVEFVKSFNLPMLMLGGGGYTIRNVARCWYETAVALDTEIPNELPYNDYF
        EYFGPDFKLHISPSNMTNQNTNEYLEKIKQRLFENLRMLPHAPGVQMQAIPEDAIPESGDEDEDDPKRISICSSDKRIACEEEFS
        DSEEEGEGGRKNSSNFKKAKRVKTEDEKEKDPEEKKEVTEEEKTKEEKPEAKGVKEEVKLA</Hsp_midline>
      </Hsp>
    </Hit_hsps>
  </Hit>
<Hit>

```

```
<Hit_num>2</Hit_num>
<Hit_id>sp|009106.1|</Hit_id>
<Hit_def>RecName: Full=Histone deacetylase 1; Short=HD1 [Mus musculus]</Hit_def>
<Hit_accession>009106</Hit_accession>
<Hit_len>482</Hit_len>
<Hit_hsps>
  <Hsp>
    <Hsp_num>1</Hsp_num>
  ...

```

## There are many other uses for Biopython

---

- reading multiple sequence alignments
- searching on remote biological sequence databases
- working with protein structure (requires numpy to be installed)
- biochemical pathways (KEGG)
- drawing pictures of genome and sequence features
- population genetics

## Why use biopython

---

Massive time saver once you know your way around the classes.

Reuse someone else's code. Very quick parsing of many common file formats.

Clean code.