

# Introduction to Biopython

Iddo Friedberg

Associate Professor

College of Veterinary Medicine

(based on a slides by Stuart Brown, NYU)



# Learning Goals

---

- Biopython as a toolkit
- Seq objects and their methods
- SeqRecord objects have data fields
- SeqIO to read and write sequence objects
- Direct access to GenBank with [Entrez.efetch](#)
- Working with BLAST results

# Modules

- Python functions are divided into three sets
  - A small core set that are always available
  - Some built-in modules such as **math** and **os** that can be imported from the basic install (ie. `>>> import math`)
  - A number of optional modules that must be downloaded and installed before you can import them: code that uses such modules is said to have “dependencies”
  - Most are available in different Linux distributions, or via pypy.org using pip (the Python Package Index)
- Anyone can write new Python modules, and often several different modules are available that can do the same task

# Download a file

- `urllib()` is a module that lets Python download files from the internet with the `.urlretrieve` method

```
>>> import urllib
```

```
>>>urllib.urlretrieve('https://raw.githubusercontent.com/biopython/biopython/master/Tests/GenBank/NC_005816.fna', 'yp.fasta')
```



- Biopython is an integrated collection of modules for “biological computation” including tools for working with DNA/protein sequences, sequence alignments, population genetics, phylogenetics, molecular structures, etc.
- It also provides interfaces to common biological databases (e.g. GenBank) and to some common locally installed software (e.g. BLAST)

# Biopython Tutorial

- Biopython has a “Tutorial & Cookbook” :  
<http://biopython.org/DIST/docs/tutorial/Tutorial.html>

by: Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck,  
Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, Bartek Wilczyński

from which, most of the following examples are drawn

# Object Oriented Code

- Python implements object oriented programming
- Classes bundle data and functionality

```
magic method" class MyClass:
    """A simple example class"""
    def __init__(self):
        self.data = []
private → self._priv = "fuggetaboutit"

    def say_hi(self):
        return 'hello world'
    def __str__(self):
        return "yoohoo"+str(self.data[:3]) + "..."
```

---

```
z = MyClass() ← instantiation
z.data = [1,"foo","bar",-5,9]
print(z) #???
```

# The Seq object

- The Seq object class is simple and fundamental for a lot of Biopython work. A Seq object can contain DNA, RNA, or protein.
- It contains a string and a defined alphabet for that string.
- The alphabets are actually defined objects such as

`IUPACAmbiguousDNA` or `IUPACProtein`

- Which are defined in the `Bio.Alphabet` module
- A Seq object with a DNA alphabet has some different methods than one with an Amino Acid alphabet

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> my_seq = Seq('AGTACACTGGT', IUPAC.unambiguous_dna())
>>> my_seq
Seq('AGTACACTGGT', IUPAC.unambiguous_dna())
>>> print(my_seq)
AGTACACTGGT
```

*This command creates the Seq object*



# Seq objects have string methods

- Seq objects have methods that work just like string objects
- You can get the `len()` of a Seq, slice it, and `count()` specific letters in it:

```
>>> my_seq = Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC',  
IUPAC.unambiguous_dna)
```

```
>>> len(my_seq)
```

```
32
```

```
>>> print(my_seq[6:9])
```

```
TGG
```

```
>>> my_seq.count("G")
```

```
9
```

# Turn a Seq object into a string

- Sometimes you will need to work with just the sequence string in a Seq object using a tool that is not aware of the Seq object methods
- Turn a Seq object into a string with `str()`

```
>>> my_seq
Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC',
    IUPACUnambiguousDNA())
>>> seq_string=str(my_seq)
>>> seq_string
'GATCGATGGGCCTATATAGGATCGAAAATCGC'
```

# Seq Objects have methods

- DNA Seq objects can `.translate()` to protein
  - With optional translation `table` and `to_stop=True` parameters

```
>>> coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG",
IUPAC.unambiguous_dna)
>>> coding_dna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
>>> print(coding_dna.translate(table=2, to_stop=True))
MAIVMGRWKGAR
```

Seq objects with a DNA alphabet have the `reverse_complement()` method:

```
>>> my_seq = Seq('TTTAAAATGCGGG', IUPAC.unambiguous_dna)
>>> print(my_seq.reverse_complement())
CCCGCATTTTAAA
```

- The `Bio.SeqUtils` module has some useful methods, such as `GC()` to calculate % of G+C bases in a DNA sequence.

```
>>> from Bio.SeqUtils import GC
>>> GC(my_seq)
46.875
```

# Protein Alphabet

- You could re-define my\_seq as a protein by changing the alphabet, which will totally change the methods that will work on it.
  - ('G','A','T','C' are valid protein letters)

```
>>> from Bio.SeqUtils import molecular_weight
>>> my_seq
Seq('AGTACACTGGT', IUPACUnambiguousDNA())
>>> print(molecular_weight(my_seq))
3436.1957
```

```
>>> my_seq.alphabet = IUPAC.protein
>>> my_seq
Seq('AGTACACTGGT', IUPACProtein())
>>> print(molecular_weight(my_seq))
912.0004
```

# SeqRecord Object

- The **SeqRecord** object is like a database record (such as GenBank). It is a complex object that contains a **Seq** object, and also annotation fields or features

`.seq`

`.id`

`.name`

`.description`

`.letter_annotations`

`.annotations`

`.features`

`.dbxrefs`

# SeqRecord Example

<http://biopython.org/wiki/SeqRecord>

## Extracting information from a SeqRecord

Lets look in detail at the well annotated `SeqRecord` objects Biopython creates from a GenBank file, such as `ls_orchid.gbk`, which we'll load using the `SeqIO` module. This file contains 94 records:

```
from Bio import SeqIO
for index, record in enumerate(SeqIO.parse("ls_orchid.gbk", "genbank")):
    print("index %i, ID = %s, length %i, with %i features"
          % (index, record.id, len(record.seq), len(record.features)))
```

# SeqIO and FASTA files

- **SeqIO** is the all purpose file read/write tool for SeqRecords
  - SeqIO can read many file types: <http://biopython.org/wiki/SeqIO>
- **SeqIO** has **.read()** and **.write()** methods
  - (do not need to “open” file first)
- It can read a text file in FASTA format
- In Biopython, **fasta** is a type of SeqRecord with specific fields
  - grab the file: [NC\\_005816.fna](#), and saved it as a text file in your current directory

```
>>> from Bio import SeqIO
>>> gene = SeqIO.read("NC_005816.fna", "fasta")
>>> gene.id
'gi|45478711|ref|NC_005816.1|'
>>> gene.seq
Seq('TGTAACGAACGGTGCAATAGTGATCCACACCCAACGCCTGAAATCAGATCCAGG...CTG',
SingleLetterAlphabet())
>>> len(gene.seq)
9609
```

# Multiple FASTA Records in one file

- The FASTA format can store many sequences in one text file
- `SeqIO.parse()` reads the records one by one
- This code creates a list of SeqRecord objects:

```
>>> from Bio import SeqIO
>>> handle = open("example.fasta", "rU")
           # "handle" is a pointer to the file
>>> seq_list = list(SeqIO.parse(handle, "fasta"))
>>> handle.close()
>>> print(seq_list[0].seq)      #shows the first sequence in the list
```



# Seq-ing deeper

Seq: an object for containing sequence information. Basic sequence operations.

```
>>> from Bio.Seq import Seq
>>> dir(Seq)
['__add__', '__class__', '__contains__', '__delattr__',
 '__dict__', '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__',
 '__getitem__', '__gt__', '__hash__', '__init__',
 '__le__', '__len__', '__lt__', '__module__', '__ne__',
 '__new__', '__radd__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', '__weakref__',
 '_get_seq_str_and_check_alphabet', 'back_transcribe',
 'complement', 'count', 'endswith', 'find', 'lower',
 'lstrip', 'reverse_complement', 'rfind', 'rsplit',
 'rstrip', 'split', 'startswith', 'strip', 'tomutable',
 'tostring', 'transcribe', 'translate', 'ungap', 'upper']
```

# Seq Source Code

```
def __init__(self, data, alphabet=Alphabet.generic_alphabet):

    # Enforce string storage
    if not isinstance(data, basestring):
        raise TypeError("The sequence data given to a Seq object should "
                        "be a string (not another Seq object etc)")

    self._data = data
    self.alphabet = alphabet


def __repr__(self):
    """Returns a (truncated) representation of the sequence for debugging."""
    if len(self) > 60:
        # Shows the last three letters as it is often useful to see if there
        # is a stop codon at the end of a sequence.
        # Note total length is 54+3+3=60
        return "{0}('{1}...{2}', {3!r})".format(self.__class__.__name__,
                                                str(self)[:54],
                                                str(self)[-3:],
                                                self.alphabet)
    else:
        return '{0}({1!r}, {2!r})'.format(self.__class__.__name__,
                                          self._data,
                                          self.alphabet)
```

# Seq Source Code

```
def transcribe(self):
    base = Alphabet._get_base_alphabet(self.alphabet)
    if isinstance(base, Alphabet.ProteinAlphabet):
        raise ValueError("Proteins cannot be transcribed!")
    if isinstance(base, Alphabet.RNAAlphabet):
        raise ValueError("RNA cannot be transcribed!")

    if self.alphabet == IUPAC.unambiguous_dna:
        alphabet = IUPAC.unambiguous_rna
    elif self.alphabet == IUPAC.ambiguous_dna:
        alphabet = IUPAC.ambiguous_rna
    else:
        alphabet = Alphabet.generic_rna
    return Seq(str(self).replace('T', 'U').replace('t', 'u'), alphabet))
```

# Create your own sequence class

Base class

```
from Bio.Seq import Seq
class MySeq(Seq):
    def __repr__(self):
        if len(self) > 60:
            # Shows the last three letters as it is often useful to see if there
            # is a stop codon at the end of a sequence.
            # Note total length is 54+3+3=60
            return "{0}('{1}---{2}', {3!r})".format(self.__class__.__name__,
                                                    str(self)[:54],
                                                    str(self)[-3:],
                                                    self.alphabet)
        else:
            return '{0}({1!r}, {2!r})'.format(self.__class__.__name__,
                                              self._data,
                                              self.alphabet)
```

Derived class

Was "..."

- The new `__repr__` overrides the old `__repr__`
- Everything else remains the same!
- `MySeq` is a *derived class* of `Seq`
- Also, `MySeq` is *inherited* from `Seq`

# Direct Access to GenBank

- BioPython has modules that can directly access databases over the Internet
- The **Entrez** module uses the NCBI Efetch service
- Efetch works on many NCBI databases including protein and PubMed literature citations
- The 'gb' data type contains much more annotation information, but **rettype='fasta'** also works
- With a few tweaks, this code could be used to download a list of GenBank ID's and save them as FASTA or GenBank files:

```
>>> from Bio import Entrez
>>> Entrez.email = "you@iastate.edu"
>>> handle = Entrez.efetch(db="nucleotide", id="186972394", rettype="gb",
retmode="text")
>>> record = SeqIO.read(handle, "genbank")
```

```
>>> print(record)
```

```
ID: EU490707.1
```

```
Name: EU490707
```

```
Description: Selenipedium aequinoctiale maturase K (matK) gene, partial cds; chloroplast.
```

```
Number of features: 3
```

```
/sequence_version=1
```



*These are sub-fields of the .annotations field*

```
/source=chloroplast Selenipedium aequinoctiale
```

```
/taxonomy=['Eukaryota', 'Viridiplantae', 'Streptophyta', 'Embryophyta', 'Tracheophyta',  
'Spermatophyta', 'Magnoliophyta', 'Liliopsida', 'Asparagales', 'Orchidaceae', 'Cypripedioideae',  
'Selenipedium']
```

```
/keywords=[]
```

```
/references=[Reference(title='Phylogenetic utility of ycf1 in orchids: a plastid gene more  
variable than matK', ...), Reference(title='Direct Submission', ...)]
```

```
/accessions=['EU490707']
```

```
/data_file_division=PLN
```

```
/date=15-JAN-2009
```

```
/organism=Selenipedium aequinoctiale
```

```
/gi=186972394
```

```
Seq('ATTTTACGAACCTGTGGAAATTTTGGTTATGACAATAAATCTAGTTTAGTA...GAA',  
IUPACAmbiguousDNA())
```

# BLAST

- BioPython has several methods to work with the popular NCBI BLAST software
- **NCBIWWW.qblast()** sends queries directly to the NCBI BLAST server. The query can be a Seq object, FASTA file, or a GenBank ID.

Do this now

```
>>> from Bio.Blast import NCBIWWW
>>> from Bio import SeqIO
>>> query = SeqIO.read("test.fasta", format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", query.seq)
>>> blast_result = result_handle.read()
>>> blast_file = open("my_blast.xml", "w") # create an xml
output file
>>> blast_file.write(blast_result)
>>> blast_file.close() # tidy up
>>> result_handle.close()
```

# XML

Extensible Markup Language

Used to encode documents in a form taht is human readable and machine readable

```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>
```

## Note

To: Tove

From: Jani

## Reminder

Don't forget me this weekend!



# XML is Extensible

```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this  
</note>
```

```
<note>  
  <date>2015-09-01</date>  
  <hour>08:30</hour>  
  <to>Tove</to>  
  <from>Jani</from>  
  <body>Don't forget me this weekend!</body>  
</note>
```

## Note

To: Tove

From: Jani

## Reminder

Don't forget me this weekend!

## Note

To: Tove

From: Jani

Date: 2015-09-01 08:30

Don't forget me this weekend!

# BLAST XML

MULTISPECIES: bacteriocin sakacin-P [Lactobacillus]

Sequence ID: [WP\\_004271264.1](#) Length: 61 Number of Matches: 1

[▶ See 11 more title\(s\)](#)

Range 1: 1 to 61 [GenPept](#) [Graphics](#) [▼](#) Next Match [▲](#) Previous Match

Score	Expect	Method	Identities	Positives	Gaps
122 bits(307)	1e-35	Compositional matrix adjust.	61/61(100%)	61/61(100%)	0/61(0%)
Query 1	MEKFIELSLKEVTAITGGKYYGNGVHCGKHSCTVDWGTAINIGNNAAANWATGGNAGWN				60
	MEKFIELSLKEVTAITGGKYYGNGVHCGKHSCTVDWGTAINIGNNAAANWATGGNAGWN				60
Sbjct 1	MEKFIELSLKEVTAITGGKYYGNGVHCGKHSCTVDWGTAINIGNNAAANWATGGNAGWN				60
Query 61	K	61			
	K				
Sbjct 61	K	61			

[Download](#) [▼](#) [GenPept](#) [Graphics](#)

MULTISPECIES: bacteriocin [Lactobacillus]

Sequence ID: [WP\\_056946897.1](#) Length: 61 Number of Matches: 1

[▶ See 3 more title\(s\)](#)

Range 1: 1 to 61 [GenPept](#) [Graphics](#) [▼](#) Next Match [▲](#) Previous Match

Score	Expect	Method	Identities	Positives	Gaps
120 bits(302)	8e-35	Compositional matrix adjust.	60/61(98%)	61/61(100%)	0/61(0%)
Query 1	MEKFIELSLKEVTAITGGKYYGNGVHCGKHSCTVDWGTAINIGNNAAANWATGGNAGWN				60
	MEKFIELSLKEVTAITGGKYYGNGVHCGK+SCTVDWGTAINIGNNAAANWATGGNAGWN				60
Sbjct 1	MEKFIELSLKEVTAITGGKYYGNGVHCGKYSCTVDWGTAINIGNNAAANWATGGNAGWN				60
Query 61	K	61			
	K				
Sbjct 61	K	61			

[Download](#) [▼](#) [GenPept](#) [Graphics](#)

bacteriocin mundticin [Enterococcus pallens]

Sequence ID: [WP\\_010758561.1](#) Length: 62 Number of Matches: 1

[▶ See 3 more title\(s\)](#)

Range 1: 1 to 61 [GenPept](#) [Graphics](#) [▼](#) Next Match [▲](#) Previous Match

Score	Expect	Method	Identities	Positives	Gaps
91.3 bits(225)	4e-23	Compositional matrix adjust.	45/61(74%)	50/61(81%)	0/61(0%)
Query 1	MEKFIELSLKEVTAITGGKYYGNGVHCGKHSCTVDWGTAINIGNNAAANWATGGNAGWN				60
	M+ LS KE+ ITGGKYYGNG+ CGK+SC+VDWG AIG IGNNAAANWATGG AGWN				60
Sbjct 1	MQNIKALSAKELIEITGGKYYGNGLSCGKYSCVDWGKAIGIIGNNAAANWATGGAAGWN				60
Query 61	K	61			
	K				
Sbjct 61	K	61			

# BLAST XML

```
<Hit>
  <Hit_num>3</Hit_num>
  <Hit_id>gi|498453073|ref|WP_010758561.1|</Hit_id>
  <Hit_def>bacteriocin mundtacin [Enterococcus pallens] &gt;gi|486858092|gb|E0H91048.1| bacteriocin mundtacin [Enterococcus pallens ATCC BAA-351] &gt;gi|508242815|gb|E0U16245.1| bacteriocin mundtacin [Enterococcus pallens ATCC BAA-351] &gt;gi|1105345489|gb|OJG79015.1| bacteriocin mundtacin [Enterococcus pallens]</Hit_def>
  <Hit_accession>WP_010758561</Hit_accession>
  <Hit_len>62</Hit_len>
  <Hit_hsps>
    <Hsp>
      <Hsp_num>1</Hsp_num>
      <Hsp_bit-score>91.2781</Hsp_bit-score>
      <Hsp_score>225</Hsp_score>
      <Hsp_evalue>4.38515e-23</Hsp_evalue>
      <Hsp_query-from>1</Hsp_query-from>
      <Hsp_query-to>61</Hsp_query-to>
      <Hsp_hit-from>1</Hsp_hit-from>
      <Hsp_hit-to>61</Hsp_hit-to>
      <Hsp_query-frame>0</Hsp_query-frame>
      <Hsp_hit-frame>0</Hsp_hit-frame>
      <Hsp_identity>45</Hsp_identity>
      <Hsp_positive>50</Hsp_positive>
      <Hsp_gaps>0</Hsp_gaps>
      <Hsp_align-len>61</Hsp_align-len>
      <Hsp_qseq>MEKFIELSLKEVTAITGGKYYGNGVHCGKHSCTVDWGT AIGNIGNNAAANWATGGNAGWNK</Hsp_qseq>
      <Hsp_hseq>MQNIKALSAKELIEITGGKYYGNLSCGKYSCSVDWGKAIGIIGNNAAANWATGGAAGWNK</Hsp_hseq>
      <Hsp_midline>M+   LS KE+  ITGGKYYGNG+ CGK+SC+VDWG AIG IGNNAAANWATGG AGWNK</Hsp_midline>
    </Hsp>
  </Hit_hsps>
</Hit>
```

bacteriocin mundtacin [Enterococcus pallens]

Sequence ID: [WP\\_010758561.1](#) Length: 62 Number of Matches: 1

[▶ See 3 more title\(s\)](#)

Range 1: 1 to 61 [GenPept](#) [Graphics](#)

▼ Next Match ▲ Previous Match

Score	Expect	Method	Identities	Positives	Gaps
91.3 bits(225)	4e-23	Compositional matrix adjust.	45/61(74%)	50/61(81%)	0/61(0%)

```
Query 1 MEKFIELSLKEVTAITGGKYYGNGVHCGKHSCTVDWGT AIGNIGNNAAANWATGGNAGWN 60
      M+   LS KE+  ITGGKYYGNG+ CGK+SC+VDWG AIG IGNNAAANWATGG AGWN
Sbjct 1 MQNIKALSAKELIEITGGKYYGNLSCGKYSCSVDWGKAIGIIGNNAAANWATGGAAGWN 60
```

```
Query 61 K 61
```

```
Sbjct 61 K 61
```

# BLAST

- BioPython has several methods to work with the popular NCBI BLAST software
- **NCBIWWW.qblast()** sends queries directly to the NCBI BLAST server. The query can be a Seq object, FASTA file, or a GenBank ID.

Are we  
there yet?

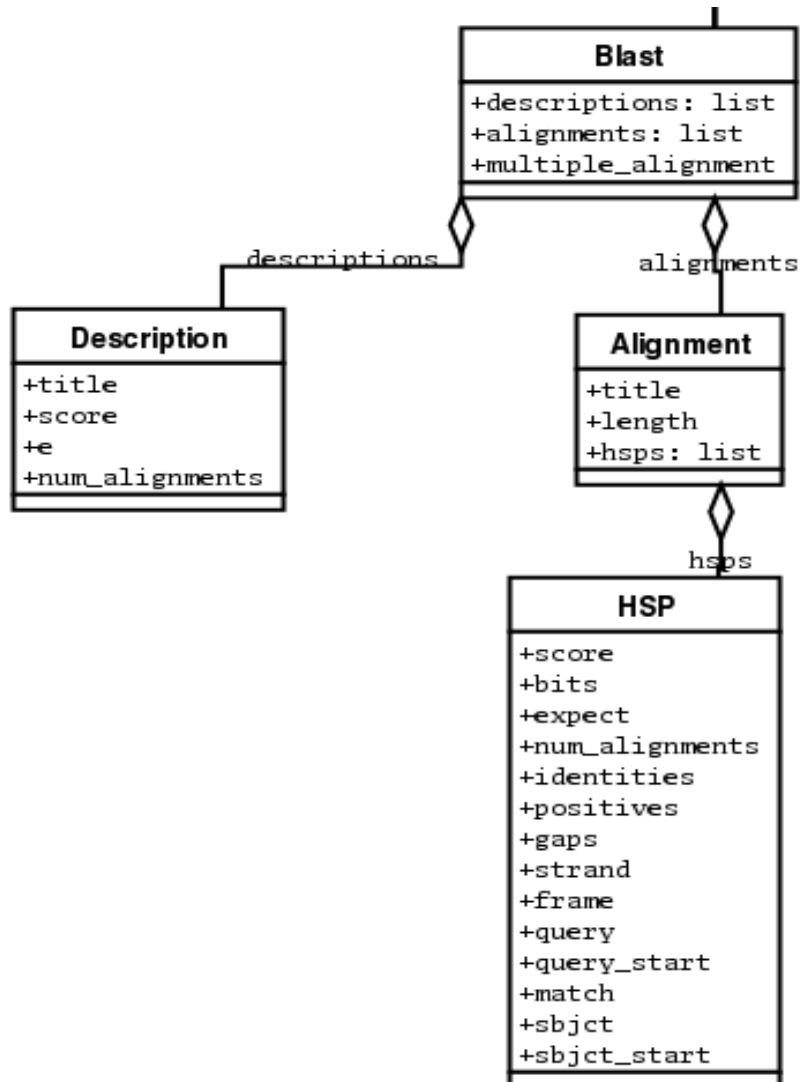
```
>>> from Bio.Blast import NCBIWWW
>>> from Bio import SeqIO
>>> query = SeqIO.read("test.fasta", format="fasta")
>>> result_handle = NCBIWWW.qblast("blastn", "nt", query.seq)
>>> blast_result = result_handle.read()
>>> blast_file = open("my_blast.xml", "w") # create an xml
output file
>>> blast_file.write(blast_result)
>>> blast_file.close() # tidy up
>>> result_handle.close()
```

# Parse BLAST Results

- It is often useful to obtain a BLAST result directly (local BLAST server or via Web browser) and then parse the result file with Python.
- Save the BLAST result in XML format
  - `NCBIXML.read()` for a file with a single BLAST result (single query)
  - `NCBIXML.parse()` for a file with multiple BLAST results (multiple queries)

```
>>> from Bio.Blast import NCBIXML
>>> handle = open("my_blast.xml")
>>> blast_record = NCBIXML.read(handle)
>>> for hit in blast_record.descriptions:
...     print(hit.title)
...     print(hit.e)
>>> handle.close()
```

# BLAST Record Object



# View Aligned Sequence

```
>>> from Bio.Blast import NCBIXML
>>> handle = open("my_blast.xml")
>>> blast_record = NCBIXML.read(handle)
>>> for hit in blast_record.alignments:
    for hsp in hit.hsps:
        print hit.title
        print hsp.expect
        print (hsp.query[0:75] + '...')
        print(hsp.match[0:75] + '...')
        print(hsp.sbjct[0:75] + '...')
```

```
gi|731383573|ref|XM_002284686.2| PREDICTED: Vitis vinifera cold-regulated 413
plasma membrane protein 2 (LOC100248690), mRNA
2.5739e-53
ATGCTAGTATGCTCGGTCATTACGGGTTTGGCACT-CATTTCTCAAATGGCTCGCCTGCCTTGCGGCTATTTAC...
|||| | || ||| ||| |  || ||||| ||||| | | ||| | || | |||| | ...
ATGCCATTAAGCTTGGTGGTCTGGGCTTTGGCACTACATTTCTTGAG-TGGTTGGCTTCTTTTGCTGCCATTTAT...
```

# Many Matches

- Often a BLAST search will return many matches for a single query (save as an XML format file)
- `NCBIXML.parse()` can return these as BLAST record objects in a list, or deal with them directly in a `for` loop.

```
from Bio.Blast import NCBIXML
E_VALUE_THRESH = 1e-20
for record in NCBIXML.parse(open("my_blast.xml")):
    if record.alignments : #skip queries with no matches
        print("QUERY: %s" % record.query[:60])
    for align in record.alignments:
        for hsp in align.hsps:
            if hsp.expect < E_VALUE_THRESH:
                print("MATCH: %s " % align.title[:60])
                print(hsp.expect)
```



# Get a file by FTP in Python

```
>>> from ftplib import FTP
```

```
>>> host="ftp.sra.ebi.ac.uk"
```

```
>>> ftp=FTP(host)
```

```
>>> ftp.login()
```

```
'230 Login successful.'
```

```
ftp.cwd('vol1/fastq/SRR020/SRR020192')
```

```
'250 Directory successfully changed.'
```

```
>>> ftp.retrlines('LIST')
```

```
-r--r--r--  1 ftp    ftp      1777817 Jun 24 20:12 SRR020192.fastq.gz
```

```
'226 Directory send OK.'
```

```
>>> ftp.retrbinary('RETR SRR020192.fastq.gz', open('SRR020192.fastq.gz',  
'wb').write)
```

```
'226 Transfer complete.'
```

```
>>> ftp.quit()
```

```
'221 Goodbye.'
```

# Learning Objectives

---

- Biopython is a toolkit
- Seq objects and their methods
- SeqRecord objects have data fields
- SeqIO to read and write sequence objects
- Direct access to GenBank with [Entrez.efetch](#)
- Working with BLAST results