National Institute of Allergy and Infectious Diseases

Introduction to Biopython Programming

R. Burke Squires
Computational Genomics Specialist
Bioinformatics and Computational Biosciences Branch (BCBB)

NIAID

National Institute of Allergy and Infectious Diseases

Welcome!

National Institute of Allergy and Infectious Diseases

NIAID

Bioinformatics & Computational Biology Branch (BCBB)

National Institute of Allergy and Infectious Diseases

NIAID

Goals

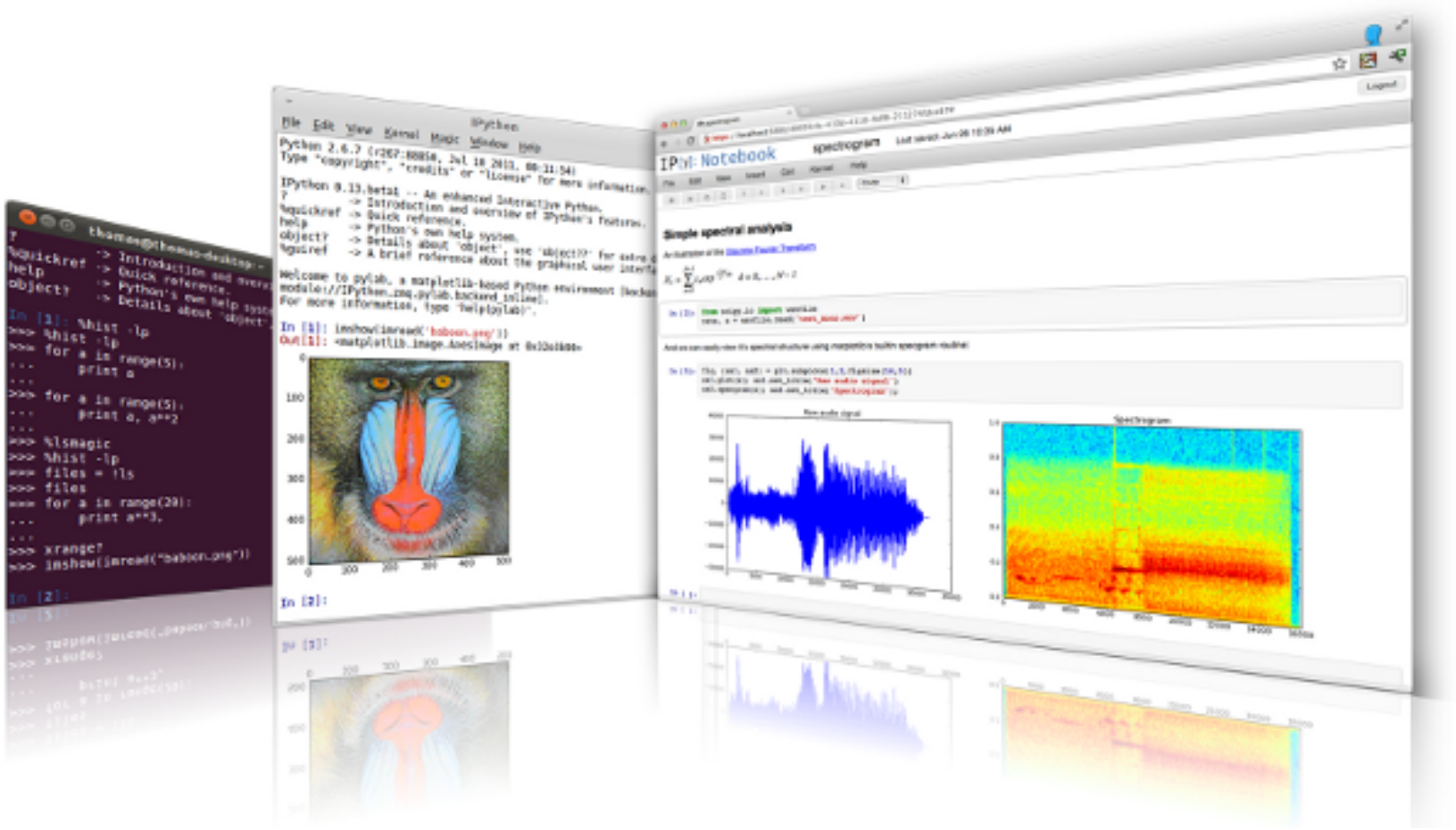
- Introduce you to the basics of the Biopython package and some of the more popular Biopython modules
- Enable you to find the information you need about Biopython
- Demonstrate how to apply Biopython to next-generation sequences data preparation
- Enable you to write or assemble scripts of your own or modify existing scripts for your own purposes
- Introduce you to EMBOSS software suite and ways to extend python and Biopython utilizing it.

Outline

- iPython Overview
- Where do you get Biopython?
- What is Biopython?
- Biopython modules
  - SeqIO
  - AlignIO
  - Additional modules
- Additional Resources
  - EMBOSS software suite
  - How do you learn more about python and Biopython?

iPython

iPython



National Institute of Allergy and Infectious Diseases

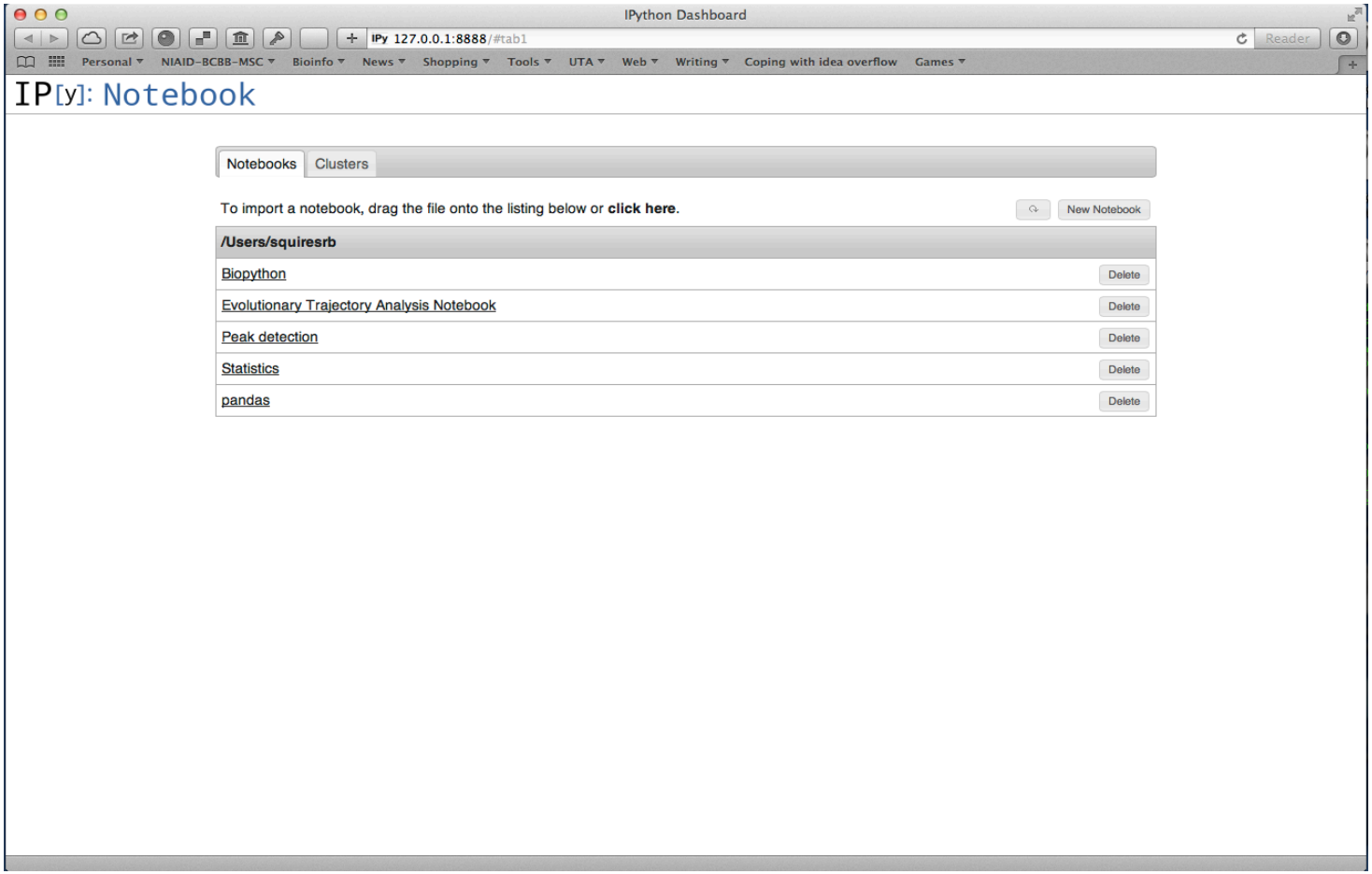
NIAID

iPython

- Already installed
  - Source: Continuum Analytics Anaconda
    - http://continuum.io/downloads.html
- Double-click on icon on desktop: 

- Launch the ipython-notebook

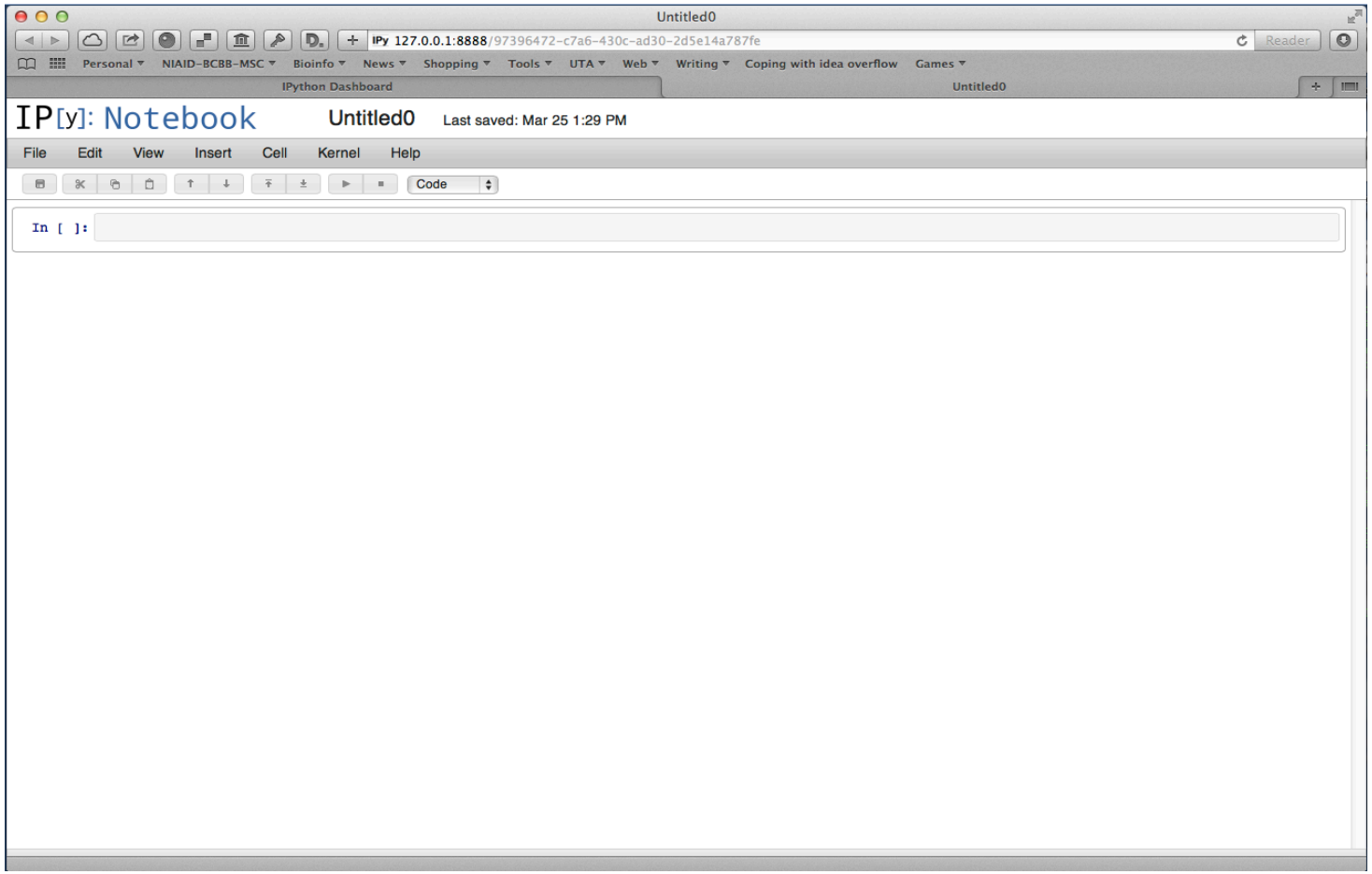
iPython – Home Screen



National Institute of Allergy and Infectious Diseases

NIAID

iPython – New Notebook



National Institute of Allergy and Infectious Diseases

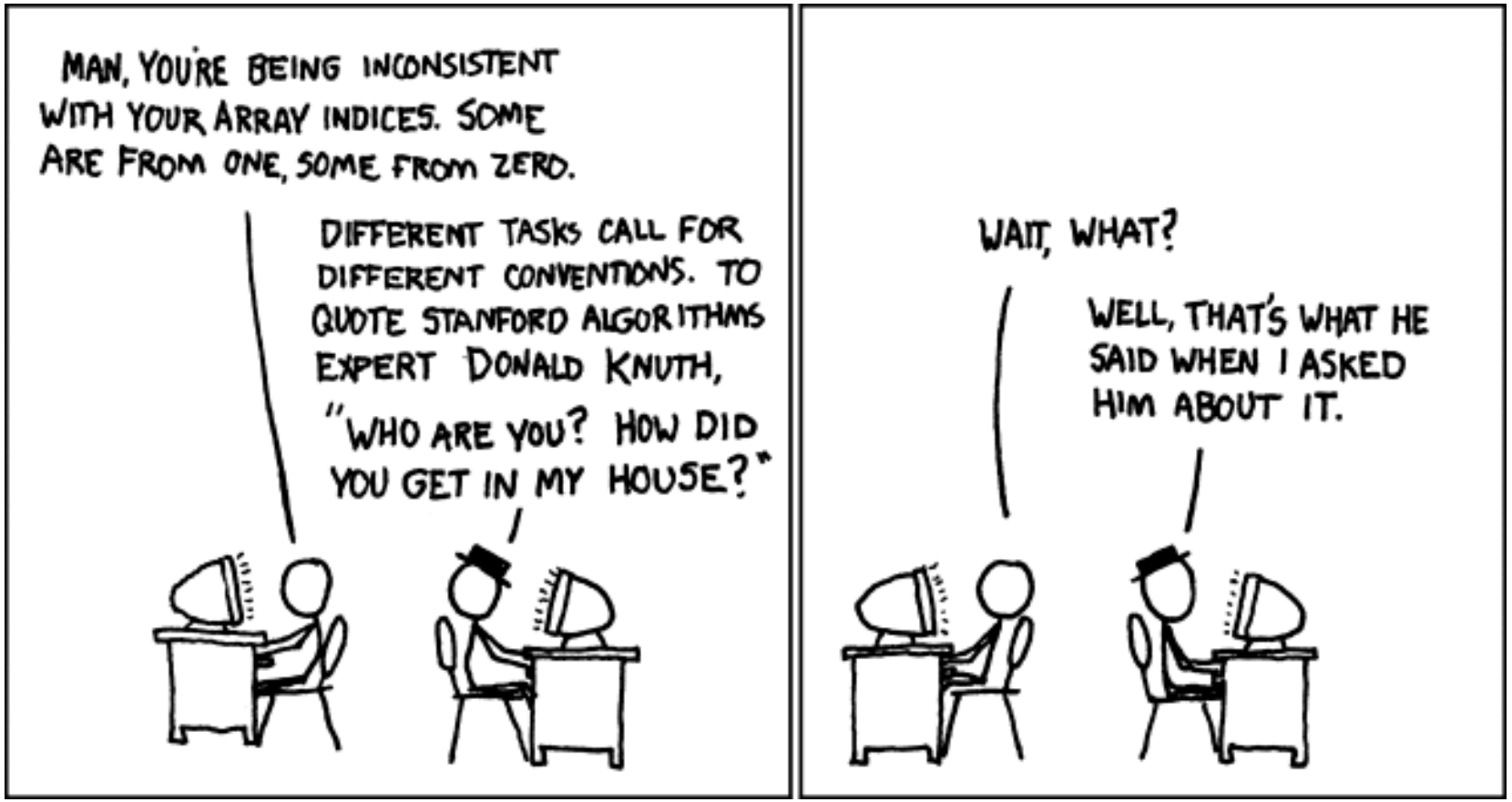
NIAID

iPython – New Notebook

- User terminal commands in iPython
  - ls – to list the contents of the current directory
  - pwd – print working directory
  - cd – change directory

- Lets change the directory to the Biopython-files folder on the desktop
  - “cd /Users/username/Desktop/biopython_files”
  - (Find user name from the pwd command above)

Indexes…Start With 0 or 1



Biopython

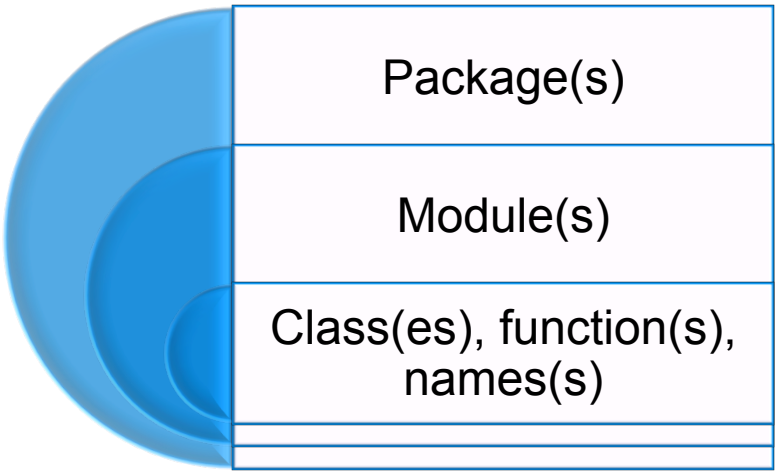
What is Biopython?

- Free, open source library for bioinformatics
- Supported by Open Bioinformatics Foundation (OBF)
  - Along with BioPerl, BioJava, BioSQL
- Runs on Mac OS X, Windows, Linux, etc.
- International team of volunteer developers
- Currently about three releases per year
- Extensive "Biopython Tutorial & Cookbook"

Python Package & Modules

- Python package: a directory of Python module(s)

- Python module: a source code file (.py) which contains:
  - Classes
  - Functions
  - Global names (variables)



Where Can I Find Details on Modules?

- In iPython or Python Interactive Shell

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

- All Biopython modules
  - http://Biopython.org/DIST/docs/api/
  - “API” application programmers interface

Jumping into Biopython…

Bio – SeqIO: Counting Records

- Count protein sequences in FASTA amino acid file, “NC_000913.faa”
- Can use “grep” to count the number of proteins
  - $ grep -c "^>" NC_000913.faa
  - 4141
- Now let's count the records with Biopython using the "SeqIO.parse" function
- Saved as “count_fasta.py” in workshop folder

Bio - SeqIO
Count Records in a FASTA File

```
from Bio import SeqIO
filename = "NC_000913.faa"
count = 0
for record in SeqIO.parse(filename, "fasta"):
    count = count + 1
print("There were %s records in file %s" % (count, filename))
```

- How many records are in the file?
  - Should be 4141

- Walkthrough

National Institute of Allergy and Infectious Diseases

Adapted from: Peter Cock, et al. BioPython Workshop

NIAID

Bio - SeqIO
Count Records in a FASTA File

- Homework
  - Modify this to count the number of records in the other FASTA files, both from *E. coli* K12 and the potato genome ("PGSC_DM_v3.4_pep_representative.fasta")
  - Using “sys.argv” get the filename as a command line argument, so that you can run it like this:
    - python count_fasta_adv.py NC_000913.ffn

Bio - SeqIO
Looking at the Sequence Records

- "SeqIO.parse" function creates SeqRecord objects.
- Biopython's "SeqRecord" objects are a container holding the sequence, and any annotation about it - most importantly the identifier.

- For FASTA files, the record identifier is taken to be the first word on the ">" line -- anything after a space is *not* part of the identifier.

- This simple example prints out the record identifers and their lengths

Bio - SeqIO
Looking at the Sequence Records

```
from Bio import SeqIO
filename = "NC_000913.faa"
for record in SeqIO.parse(filename, "fasta"):
    print("Record %s, length %i" % (record.id, len(record.seq)))
```

($ python record_lengths.py)
Record gi|16127995|ref|NP_414542.1|, length 21
Record gi|16127996|ref|NP_414543.1|, length 820
Record gi|16127997|ref|NP_414544.1|, length 310
...
Record gi|16132219|ref|NP_418819.1|, length 46
Record gi|16132220|ref|NP_418820.1|, length 228

Bio - SeqIO
Looking at the Sequence Records, cont.

- Homework
  - Count how many sequences are <100 amino acids long
  - Create a modified script "total_length.py" based on the above examples which counts the number of records and calculates the total length of all the sequences (i.e. "21 + 820 + 310 + 428 + ... + 46 + 228"), giving:
    - $ python total_length.py
    - 4141 records, total length 1311442
  - Plot a histogram of the sequence length distribution (tip - see the `Biopython Tutorial & Cookbook)

Bio - SeqIO
Looking at the Sequence

- The "SeqRecord" objects the identifiers are stored as standard Python strings (e.g. ".id"). For the sequence, Biopython uses a string-like "Seq" object, accessed as ".seq".

- In many ways the "Seq" objects act like Python strings, you can print them, take their length using the "len(...)" function, and slice them with square brackets to get a sub-sequence or a single letter.

Bio - SeqIO
Record Lengths

- Using "SeqIO.parse(...)" in a for loop, for each record print out the identifier, the first 10 letters of each sequences, the last 10 letters

```
from Bio import SeqIO
filename = "NC_000913.faa"
for record in SeqIO.parse(filename, "fasta"):
    start_seq = record.seq[:10] # first 10 letters
    end_seq = record.seq[-10:] # last 10 letters
    print(record.id + " " + start_seq + "..." + end_seq)
```

National Institute of Allergy and Infectious Diseases

Adapted from: Peter Cock, et al. BioPython Workshop

NIAID

Bio - SeqIO
Check for Initial Methionine

- How to check all the protein sequences start with a methionine (represented as the letter "M" in the standard IUPAC single letter amino acid code), and count how many records fail this

- python check_start_met.py
  - Found 0 records in NC_000913.faa which did not start with M

- Good - no strange proteins. This genome has been completely sequenced and a lot of work has been done on the annotation, so it is a 'Gold Standard'.

Bio - SeqIO
Check for Initial Methionine

```
from Bio import SeqIO
#filename = "NC_000913.faa"
filename = "PGSC_DM_v3.4_pep_representative.fasta"
bad = 0
for record in SeqIO.parse(filename, "fasta"):
    if not record.seq.startswith("M"):
        bad = bad + 1
        print(record.id + " starts " + record.seq[0])
print("Found " + str(bad) + " records in " + filename + " which
did not start with M")
```

National Institute of Allergy and Infectious Diseases

Adapted from: Peter Cock, et al. BioPython Workshop

NIAID

Bio - SeqIO
Check for Initial Methionine

- Now try this on the potato protein file "PGSC_DM_v3.4_pep_representative.fasta"
  - $ python check_start_met.py
  - PGSC0003DMP400032467 starts T
  - PGSC0003DMP400011427 starts Q
  - PGSC0003DMP400068739 starts E
  - ...
  - PGSC0003DMP400011481 starts Y
  - Found 208 records in PGSC_DM_v3.4_pep_representative.fasta which did not start with M
- Homework
  - Modify this script to print out the description of the problem records, not just the identifier. *Tip*: Try reading the documentation, e.g. Biopython's wiki page on the `SeqRecord <http://biopython.org/wiki/SeqRecord>`.
- Discussion: What did you notice about these record descriptions? Can you think of any reasons why there could be so many genes/proteins with a problem at the start?

Bio - SeqIO
Check for Stop Codons

- Let's check the example protein FASTA files for any "*" symbols in the sequence. For this you can use several of the standard Python string operations which also apply to "Seq" objects
  - my_string = "MLNTCRVPLTDRKVKEKRAMKQHKAMIVALIVICITAVVAALVTRKDLCEV HIRTGQTEVAVFTAYESE*”
  - my_string.startswith("M")
    - True
  - my_string.endswith("*")
    - True
  - len(my_string)
    - 70
  - my_string.count("M")
    - 3
  - my_string.count("*")
    - 1

Bio - SeqIO
Check for Stop Codons

```
from Bio import SeqIO
filename = "NC_000913.faa"
#filename = "PGSC_DM_v3.4_pep_representative.fasta"
contains_star = 0
ends_with_star = 0
print("Checking " + filename + " for terminal stop codons")
for record in SeqIO.parse(filename, "fasta"):
    if record.seq.count("*"):
        contains_star = contains_star + 1
    if record.seq.endswith("*"):
        ends_with_star = ends_with_star + 1
print(str(contains_star) + " records with * in them")
print(str(ends_with_star) + " with * at the end")
```

National Institute of Allergy and Infectious Diseases

Adapted from: Peter Cock, et al. BioPython Workshop

NIAID

Bio - SeqIO
Different File Formats

- If you work with finished genomes, you'll often see nicely annotated files in the EMBL or GenBank format. Let's try this with the *E. coli* K12 GenBank file, "NC_000913.gbk", based on the previous example:

```
from Bio import SeqIO
fasta_record = SeqIO.read("NC_000913.fna", "fasta”)
print(fasta_record.id + " length " + str(len(fasta_record)))

gi|556503834|ref|NC_000913.3| length 4641652

genbank_record = SeqIO.read("NC_000913.gbk", "genbank”)
print(genbank_record.id + " length " + str(len(genbank_record)))

NC_000913.3 length 4641652
```

Writing Sequence Files in Biopython

Bio – SeqIO: Converting a Sequence File

- Recall we looked at the *E. coli* K12 chromosome as a FASTA file "NC_000913.fna" and as a GenBank file "NC_000913.gbk". Suppose we only had the GenBank file, and wanted to turn it into a FASTA file?

- Biopython's "SeqIO" module can read and write lots of sequence file formats, and has a handy helper function to convert a file.

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

Bio – SeqIO: Converting a Sequence File

```
from Bio import SeqIO
input_filename = "NC_000913.gbk"
output_filename = "NC_000913_converted.fasta"
count = SeqIO.convert(input_filename, "gb", output_filename,
"fasta")
print(str(count) + " records converted")
```

- Homework
  - Modify this to add command line parsing to take the input and output filenames as arguments

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Bio – SeqIO: Filtering a Sequence File

- Suppose we wanted to filter a FASTA file by length, for example exclude protein sequences less than 100 amino acids long.

```
from Bio import SeqIO
input_filename = "NC_000913.faa"
output_filename = "NC_000913_long_only.faa"
count = 0
total = 0
output_handle = open(output_filename, "w")
for record in SeqIO.parse(input_filename, "fasta"):
    total = total + 1
    if 100 <= len(record):
        count = count + 1
        SeqIO.write(record, output_handle, "fasta")
output_handle.close()
print(str(count) + " records selected out of " + str(total))
```

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Bio – SeqIO: Editing Sequences

- Previous examples had a terminal "*" character (stop codon). Python strings, Biopython "Seq" and "SeqRecord" objects can all be *sliced* to extract a sub-sequence or partial record. In this case, we want to take everything up to but excluding the final letter:

```
my_seq =
"MTAIVIGAKILGIIYSSPQLRKCNSATQNDHSDLQISFWKDHLRQCTTNS*"
cut_seq = my_seq[:-1] # remove last letter
print(cut_seq)
MTAIVIGAKILGIIYSSPQLRKCNSATQNDHSDLQISFWKDHLRQCTTNS
```

- Homework
  - Modify the following example to only remove the last letter if it is a "*" (and save the original record unchanged if it does not end with "*").

Bio – SeqIO: Editing Sequences

```
from Bio import SeqIO
input_filename = "PGSC_DM_v3.4_pep_representative.fasta"
output_filename = "PGSC_DM_v3.4_pep_rep_no_stars.fasta"
output_handle = open(output_filename, "w")
for record in SeqIO.parse(input_filename, "fasta"):
    record = record[:-1]
    SeqIO.write(record,output_handle, "fasta")
output_handle.close()
```

Sample solution is called "cut_final_star.py".

Bio – SeqIO: Filtering by Record Name

- A very common task is pulling out particular sequences from a large sequence file. Membership testing with Python lists (or sets) is one neat way to do this.

- Write a new script starting as follows which writes out the potato proteins on this list

Bio – SeqIO:
Filtering by Record Name

Solution: filter_wanted_ids.py

```
from Bio import SeqIO
wanted_ids = ["PGSC0003DMP400019313", "PGSC0003DMP400020381",
"PGSC0003DMP400020972"]
input_filename = "PGSC_DM_v3.4_pep_representative.fasta"
output_filename = "wanted_potato_proteins.fasta"
count = 0
total = 0
output_handle = open(output_filename, "w")
for record in SeqIO.parse(input_filename, "fasta"):
    total = total + 1
    if record.id in wanted_ids:
        count = count + 1
        SeqIO.write(record, output_handle, "fasta")
output_handle.close()
print(str(count) + " records selected out of " + str(total))
```

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

Bio – SeqIO: Filtering by Record Name

- Advanced Exercise
  - Modify this to read the list of wanted identifiers from a plain text input file (one identifier per line).

- Discussion
  - What happens if a wanted identifier is not in the input file?
  - What happens if an identifier appears twice?
  - What order is the output file?

Bio – SeqIO: Selecting by Record Name

- What if you want the records in the specified order (regardless of the order in the FASTA file)?
- In this situation, you can't make a single for loop over the FASTA file. For a tiny file you could load everything into memory (e.g. as a Python dictionary), but that won't work on larger files.
- Instead, we can use Biopython's "SeqIO.index(...)" function which lets us treat a sequence file like a Python dictionary.

Bio – SeqIO:
Selecting by Record Name

```
from Bio import SeqIO
filename = "PGSC_DM_v3.4_pep_representative.fasta"
fasta_index = SeqIO.index(filename, "fasta")
print(str(len(fasta_index)) + " records in " + filename)
record = fasta_index["PGSC0003DMP400019313”]
print(record)
```

```
ID: PGSC0003DMP400019313
Name: PGSC0003DMP400019313
Description: PGSC0003DMP400019313 PGSC0003DMT400028369 Protein
Number of features: 0
Seq('MSKSLYLSLFFLSFVVALFGILPNVKGNILDDICPGSFFPPLCFQMLRNDPSVS...LK
*', SingleLetterAlphabet())
```

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Bio – SeqIO: Selecting by Record Name

```
from Bio import SeqIO
wanted_ids = ["PGSC0003DMP400019313", "PGSC0003DMP400020381",
"PGSC0003DMP400020972"]
input_filename = "PGSC_DM_v3.4_pep_representative.fasta"
output_filename = "wanted_potato_proteins_in_order.fasta"
fasta_index = SeqIO.index(input_filename, "fasta")
count = 0
total = len(fasta_index)
output_handle = open(output_filename, "w")
for identifier in wanted_ids:
    record = fasta_index[identifier]
    SeqIO.write(record, output_handle, "fasta")
    count = count + 1
output_handle.close()
print(str(count) + " records selected out of " + str(total))
```

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Bio – AlignIO: Reading Multiple-sequence Alignments

- We're going to look at a small seed alignment for one of the PFAM domains, the `A2L zinc ribbon domain (A2L_zn_ribbon; PF08792). This was picked almost at random - it is small enough to see the entire alignment on screen, and has some obvious gap-rich columns.

- From the alignments tab on the Pfam webpage, you can download the raw alignment in several different formats (Selex, Stockholm, FASTA, and MSF). Biopython is able to work with FASTA (very simple) and Stockholm format (richly annotated).

Bio – AlignIO: Loading a Single Alignment

- As in "SeqIO", under "AlignIO" we have both
  - "AlignIO.parse(...)" for looping over multiple separate alignments
  - "AlignIO.read(...)" for loading a file containing a single alignment

- Most of the time you will be working with alignment files which contain a single alignment, so normally you will use "AlignIO.read(..)".

- Here is an example loading the Pfam seed alignment for the `A2L zinc ribbon domain (A2L_zn_ribbon; PF08792):

```
from Bio import AlignIO
alignment = AlignIO.read("PF08792_seed.sth", "stockholm”)
print(alignment)
```

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

Bio – AlignIO:
Loading a Single Alignment

```
SingleLetterAlphabet() alignment with 14 rows and 37 columns
SIPVVCT---CGNDKDFY--KDDDIYICQLCNAETVK VF282_IIV6/150-181
DIIENCKY--CGSFDIE---KVKDIYTCGDCTQTYTT Q9YW27_MSEPV/2-33
SDNIKCKY--CNSFNII---KNKDIYSCCDCSNCYTT Q9EMK1_AMEPV/2-33
AQDWRCDD--CNATLVYV--KKDAQRVCLECGKSTFF Q6XM16_9PHYC/83-115
SKEWICEV--CNKELVYI--RKDAERVCPDCGLSHPY Q8QNH7_ESV1K/101-133
NDDSKCIK--CGGPVLMQ--AARSLLNCQECGYSAAV Q4A276_EHV8U/148-180
KSQNVCSVPDCDGEKILN--QNDGYMVCKKCGFSEPI YR429_MIMIV/213-247
LKYKECKY--CHTDMVFN--TTQFGLQCPNCGCIQEL VF385_ASFB7/145-177
RNLKSCSN--CKHNGLI---TEYNHEFCIFCQSVFQL Q6VZA9_CNPV/2-33
MNLRMCGG--CRRNGLV---SDADYEFCLFCETVFPM Q6TVP3_ORFSA/1-32
MNLRLCSG--CRHNGIV---SEQGYEYCIFCESVFQK VLTF3_VACCC/1-32
MNLKMCSG--CSHNGIV---SEHGYEFCIFCESIFQS Q8V3K7_SWPV1/1-32
NALRHCHG--CKHNGLV---LEQGYEFCIFCQAVFQH O11357_MCV1/5-36
DQIYTCT---CGGQMELWVNSTQSDLVCNECGATQPY Y494R_PBCV1/148-181
```

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Bio – AlignIO: Loading a Single Alignment

- In many ways, the alignment acts like a list of "SeqRecord" objects (just like you would get from "SeqIO").

```
print(len(alignment))
14
```

- The length of the alignment is the number of rows for example, and you can loop over the rows as individual "SeqRecord" objects:

```
for record in alignment:
    print(record.id + " has " + str(record.seq.count("-")) + "
gaps")
```

VF282_IIV6/150-181 has 5 gaps
Q9YW27_MSEPV/2-33 has 5 gaps
Q9EMK1_AMEPV/2-33 has 5 gaps
Q6XM16_9PHYC/83-115 has 4 gaps

Bio – AlignIO: Loading a Single Alignment

- Homework
  - Write a python script called "count_gaps.py" which reports the number or records, the total number of gaps, and the mean (average) number of gaps per record:

Bio – AlignIO: Writing Multiple-sequence Alignment Files

- As you might guess from using "SeqIO.convert(...)" and "SeqIO.write(...)", there are matching "AlignIO.convert()" and "AlignIO.write(...)" functions.

- For example, this will convert the Stockholm formatted alignment into a relaxed PHYLIP format file:

```
from Bio import AlignIO
input_filename = "PF08792_seed.sth"
output_filename = "PF08792_seed_converted.phy"
AlignIO.convert(input_filename, "stockholm",
output_filename, "phylip-relaxed")
```

- Homework
  - Modify this example to convert the Stockholm file into a FASTA alignment file.

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Bio – AlignIO: Writing Multiple-sequence Alignment Files

- This "AlignIO.convert(...)" example is equivalent to using "AlignIO.read(...)" and "AlignIO.write(...)" explicitly:

```
from Bio import AlignIO
input_filename = "PF08792_seed.sth”
output_filename = "PF08792_seed_converted.phy”
alignment = AlignIO.read(input_filename, "stockholm”)
AlignIO.write(alignment, output_filename, "phylip-relaxed")
```

- This form is most useful if you wish to modify the alignment in some way, which we will do next.

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Bio – AlignIO: Sorting the Rows

- How you can sort the rows by identifier within Biopython:

```
from Bio import AlignIO
alignment = AlignIO.read("PF08792_seed.sth", "stockholm”)
alignment.sort()
print(alignment)
```

```
SingleLetterAlphabet() alignment with 14 rows and 37 columns
NALRHCHG--CKHNGLV---LEQGYEFCIFCQAVFQH O11357_MCV1/5-36
NDDSKCIK--CGGPVLMQ--AARSLLNCQECGYSAAV Q4A276_EHV8U/148-180
MNLRMCGG--CRRNGLV---SDADYEFCLFCETVFPM Q6TVP3_ORFSA/1-32
RNLKSCSN--CKHNGLI---TEYNHEFCIFCQSVFQL Q6VZA9_CNPV/2-33
AQDWRCDD--CNATLVYV--KKDAQRVCLECGKSTFF Q6XM16_9PHYC/83-115
```

Bio – AlignIO: Sorting the Rows

- Homework
  - Write a Python script "sort_alignment_by_id.py" which uses "AlignIO.read(..)" and "AlignIO.write(..)" to convert "PF08792_seed.sth" into a sorted FASTA file.

- By default the alignment's sort method uses the identifers as the sort key, but much like how sorting a Python list works, you can override this.

- Homework
  - Define your own function taking a single argument (a "SeqRecord") which returns the number of gaps in the sequence. Use this to sort the alignment and print it to screen (or save it as a new file)

Sequence Features

Sequence Features: Working with Sequence Features

- Most of the time GenBank files contain a single record for a single chromosome or plasmid, so we'll generally use the "SeqIO.read(...)" function. Remember the second argument is the file format, so if we start from the code to read in a FASTA file

Sequence Features:
Working with Sequence Features

```
from Bio import SeqIO
record = SeqIO.read("NC_000913.fna", "fasta")
print(record.id)
gi|556503834|ref|NC_000913.3|
print(len(record))
4641652
print(len(record.features))
0

from Bio import SeqIO
record = SeqIO.read("NC_000913.gbk", "genbank")
print(record.id)
NC_000913.3
print(len(record))
4641652
print(len(record.features))
23086
```

Sequence Features:
Working with Sequence Features

```
my_gene = record.features[3]
print(my_gene)
type: gene
location: [336:2799](+)
qualifiers:
    Key: db_xref, Value: ['EcoGene:EG10998', 'GeneID:945803']
    Key: gene, Value: ['thrA']
    Key: gene_synonym, Value: ['ECK0002; Hs; JW0001; thrA1;
thrA2; thrD']
    Key: locus_tag, Value: ['b0002']
```

National Institute of Allergy and Infectious Diseases

Source: Peter Cock, et al. BioPython Workshop

NIAID

Sequence Features:
Working with Sequence Features

- Doing a print like this tries to give a human readable display. There are three key properties:
  - ".type" which is a string like "gene" or "CDS"
  - ".location" which describes where on the genome this feature is, and
  - ".qualifiers" which is a Python dictionary full of all the annotation for the feature (things like gene identifiers).

- This is what this gene looks like in the raw GenBank file::

```
gene     337..2799
         /gene="thrA”
         /locus_tag="b0002
         /gene_synonym="ECK0002; Hs; JW0001; thrA1; thrA2; thrD”
         /db_xref="EcoGene:EG10998”
         /db_xref="GeneID:945803”
```

Sequence Features: Feature Locations

- We're going to focus on using the location information for different feature types. Continuing with the same example:

```
from Bio import SeqIO
record = SeqIO.read("NC_000913.gbk", "genbank")
my_gene = record.features[3]
print(my_gene.qualifiers["locus_tag"])
['b0002']
print(my_gene.location)
[336:2799](+)
print(my_gene.location.start)
336
print(my_gene.location.end)
2799
print(my_gene.location.strand)
1
```

Extracting Info from GenBank Record

```
from Bio import SeqIO
for index, record in enumerate(SeqIO.parse(open("NC_000913.gbk"), "genbank")):
    print "index %i, ID = %s, length %i, with %i feat. " \
            % (index, record.id, len(record.seq), len(record.features))
```

- Output

```
Index 0, ID = Z78533.1, length 740, with 5 feat.
index 1, ID = Z78532.1, length 753, with 5 feat.
index 2, ID = Z78531.1, length 748, with 5 feat.
index 3, ID = Z78530.1, length 744, with 5 feat.
index 4, ID = Z78529.1, length 733, with 5 feat.
index 5, ID = Z78527.1, length 718, with 5 feat.
index 6, ID = Z78526.1, length 730, with 5 feat.
index 7, ID = Z78525.1, length 704, with 5 feat.
```

BLAST

A Few BLAST Details

Alignment starts with initial word of 11

```
ACACTGAGTGA
||||||||||
ACACTGAGTGA
```

Extension to the left has no mismatches, no penalty points
Extension to the right has mismatches and penalty points

```
GCACCTTTGCCACACTGAGTGAGCTGCTCTATG
||||||||||||||||||||| |||| || |
GCACCTTTGCCACACTGAGTGACCTGCACTGTA
```

Extension to the left has no penalty points and can continue to grow
Extension to the right accumulates too many mismatch penalty points; extension in this direction stop

```
CAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCTCTATGGTCCTTTGGGG
|||||||||||||||||||||||||||||||||| |||| || |    ||||
CAACCTCAAGGGCACCTTTGCCACACTGAGTGACCTGCACTGTAAAGTTTTGCAT
```

If left side cannot grow any more, the final alignment looks like this:

```
CAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCTGCTCTATG
|||||||||||||||||||||||||||||||||| |||| || |
CAACCTCAAGGGCACCTTTGCCACACTGAGTGACCTGCACTGTA
```

BLAST Output (Text)

BLASTN 2.2.28+
Reference: Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller (2000), "A greedy algorithm for aligning DNA sequences", J Comput Biol 2000; 7(1-2):203-14.

Reference for database indexing: Aleksandr Morgulis, George Coulouris, Yan Raytselis, Thomas L. Madden, Richa Agarwala, Alejandro A. Schaffer (2008), "Database Indexing for Production MegaBLAST Searches", Bioinformatics 24:1757-1764.

RID: SJ2EFD07014

Database: Nucleotide collection (nt)
26,000,382 sequences; 49,159,429,833 total letters
Query= gi|2765658|emb|Z78533.1| C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA

Length=740

Sequences producing significant alignments:	Score (Bits)	E Value
emb|Z78533.1| C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA	1367	0.0
emb|FR720328.1| Cypripedium irapeanum ITS1, 5.8S rRNA gene, I...	1210	0.0

National Institute of Allergy and Infectious Diseases

NIAID

BLAST Output (XML)

```
<?xml version="1.0"?>
<!DOCTYPE BlastOutput PUBLIC "-//NCBI//NCBI BlastOutput/EN" "http://www.ncbi.nlm.nih.gov/dtd/
NCBI_BlastOutput.dtd">
<BlastOutput>
  <BlastOutput_program>blastn</BlastOutput_program>
  <BlastOutput_version>BLASTN 2.2.28+</BlastOutput_version>
  <BlastOutput_reference>Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller (2000), &quot;A greedy
algorithm for aligning DNA sequences&quot;, J Comput Biol 2000; 7(1-2):203-14.</BlastOutput_reference>
  <BlastOutput_db>nr</BlastOutput_db>
  <BlastOutput_query-ID>gi|2765658|emb|Z78533.1|</BlastOutput_query-ID>
  <BlastOutput_query-def>C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA</BlastOutput_query-def>
  <BlastOutput_query-len>740</BlastOutput_query-len>
  <BlastOutput_param>
    <Parameters>
      <Parameters_expect>10</Parameters_expect>
      <Parameters_sc-match>1</Parameters_sc-match>
      <Parameters_sc-mismatch>-2</Parameters_sc-mismatch>
      <Parameters_gap-open>0</Parameters_gap-open>
      <Parameters_gap-extend>0</Parameters_gap-extend>
      <Parameters_filter>L;m;</Parameters_filter>
    </Parameters>
  </BlastOutput_param>
<BlastOutput_iterations>
<Iteration>
  <Iteration_iter-num>1</Iteration_iter-num>
  <Iteration_query-ID>gi|2765658|emb|Z78533.1|</Iteration_query-ID>
  <Iteration_query-def>C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA</Iteration_query-def>
  <Iteration_query-len>740</Iteration_query-len>
<Iteration_hits>
```

BLAST a Sequence to File

```
from Bio.Blast import NCBIWWW
from Bio import SeqIO

record = SeqIO.read(open("m_cold.fasta"), format="fasta")
result_handle = NCBIWWW.qblast("blastn", "nt", record.seq)
save_file = open("my_blast.xml", "w")
save_file.write(result_handle.read())
save_file.close()
result_handle.close()
```

National Institute of Allergy and Infectious Diseases

Source: BioPython Tutorial and Cookbook

Parse BLAST output

```
from Bio.Blast import NCBIXML
result_handle = open("my_blast.xml")
blast_record = NCBIXML.read(result_handle)
E_VALUE_THRESH = 0.04
for alignment in blast_record.alignments:
    for hsp in alignment.hsps:
        if hsp.expect < E_VALUE_THRESH:
            print '****Alignment****'
            print 'sequence:', alignment.title
            print 'length:', alignment.length
            print 'e value:', hsp.expect
            print hsp.query[0:75] + '...'
            print hsp.match[0:75] + '...'
            print hsp.sbjct[0:75] + '...'
```

Next-Gen Related Scripts

- http://Biopython.org/DIST/docs/tutorial/Tutorial.html#htoc217
- Chapter 18 Cookbook – Cool things to do with it
  - 18.1 Working with sequence files
    - 18.1.1 Filtering a sequence file
    - 18.1.2 Producing randomised genomes
    - 18.1.3 Translating a FASTA file of CDS entries
    - 18.1.4 Making the sequences in a FASTA file upper case
    - 18.1.5 Sorting a sequence file
    - 18.1.6 **Simple quality filtering for FASTQ files**
    - 18.1.7 Trimming off primer sequences
    - 18.1.8 **Trimming off adaptor sequences**
    - 18.1.9 **Converting FASTQ files**
    - 18.1.10 **Converting FASTA and QUAL files into FASTQ files**
    - 18.1.11 Indexing a FASTQ file
    - 18.1.12 Converting SFF files
    - 18.1.13 Identifying open reading frames

FASTA, QUAL <=> FASTQ Conversion

- Going from FASTQ to FASTA:

```
from Bio import SeqIO
SeqIO.convert("sample1.fq", "fastq", "sample1.fasta", "fasta")
```

- Going from FASTQ to QUAL is also easy:

```
from Bio import SeqIO
SeqIO.convert("sample1.fq", "fastq", "sample1.qual", "qual")
```

- FASTA and QUAL file to FASTQ:

```
from Bio.SeqIO.QualityIO import PairedFastaQualIterator
for record in PairedFastaQualIterator(open("example.fasta"), open("example.qual")):
  print record
```

Clean up FASTQ Files by Phred Score

- Download example data file:
  - ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR020/SRR020192/SRR020192.fastq.gz

```
from Bio import SeqIO
good_reads = (rec for rec in \
        SeqIO.parse("SRR020192.fastq", "fastq") \
        if min(rec.letter_annotations["phred_quality"]) >= 20)
count = SeqIO.write(good_reads, "good_quality.fastq", "fastq")
print "Saved %i reads" % count
```

Trimming Off Adaptor Sequences

```
from Bio import SeqIO
def trim_adaptors(records, adaptor, min_len):
len_adaptor = len(adaptor)                  # cache this for later
    for record in records:
        len_record = len(record)            # Cache this for later
        if len(record) < min_len:           # Too short to keep
           continue
        index = record.seq.find(adaptor)
        if index == -1:                 # Aadaptor not found, so won't trim
            yield record
        elif len_record - index - len_adaptor >= min_len:
            #after trimming this will still be long enough
            yield record[index+len_adaptor:]

original_reads = SeqIO.parse("SRR020192.fastq", "fastq")
trimmed_reads = trim_adaptors(original_reads, "GATGACGGTGT", 100)
count = SeqIO.write(trimmed_reads, "trimmed.fastq", "fastq")
print "Saved %i reads" % count
```

Biopython BioSQL Interface

- Can use MySQL or Postgres Database
- Must install and setup database software, database
- Must load data into database
- Python scripts in BioSQL folder (within biopython folder)

Biopython BioSQL Interface

```
from BioSQL import BioSeqDatabase
server = BioSeqDatabase.open_database(driver="MySQLdb",
       user="root", passwd = "",
       host = "localhost", db="bioseqdb")
db = server["orchids"]
for identifier in ['6273291', '6273290', '6273289'] :
    seq_record = db.lookup(gi=identifier)
    print seq_record.id, \
       seq_record.description[:50] + "..."
    print "Sequence length %i," % len(seq_record.seq)
```

Additional Resources

EMBOSS

- European Molecular Biology Open Source Suite
  - http://emboss.sourceforge.net
- Command line programs to accomplish many bioinformatics tasks
- Try out (for NIH access)
  - http://helixweb.nih.gov/emboss/
- Biopython supports through Bio.EMBOSS
  - http://Biopython.org/DIST/docs/api/Bio.Emboss-module.html

Resources: Python Programming

- Websites
  - http://wiki.python.org/moin/BeginnersGuide/NonProgrammers
  - http://www.pythonforbeginners.com
- Free eBook in HTML / PDF
  - http://greenteapress.com/thinkpython/
  - http://openbookproject.net/books/bpp4awd/index.html
- Cheatsheets
  - http://www.pythonforbeginners.com/cheatsheet/python-cheat-sheets/
- Python Regular Expressions (pattern matching)
  - http://www.pythonregex.com
- Python Style Guide
  - http://www.python.org/dev/peps/pep-0008/

Goals

- Introduce you to the basics of the Biopython package and some of the more popular Biopython modules
- Enable you to find the information you need about Biopython
- Demonstrate how to apply Biopython to next-generation sequences data preparation
- Enable you to write or assemble scripts of your own or modify existing scripts for your own purposes
- Introduce you to EMBOSS software suite and ways to extend python and Biopython utilizing it.

Q & A

Collaborations welcome

One-on-one training available (for those on NIH campus and related agencies)

ScienceApps@niaid.nih.gov

National Institute of Allergy and Infectious Diseases

NIAID