Week 12 - Lecture 23

István Albert
Huck Institutes for the Life Sciences
Tapping into data sources
Entrez: Cross-Database Search System
Entrez Global Query

- Entrez Global Query access to all databases simultaneously with a single query string and user interface.

  - nucleotide database → GenBank
  - microarray database → GEO
  - small molecule chemical structures → PubChem
  - ... very large number of other databases

Scientific communities also maintain separate resources for organisms of interest – those are the most up to date
Biopython: http://biopython.org

Introduction

Biopython is a set of freely available tools for biological computation written in Python by an international team of developers.

It is a distributed collaborative effort to develop Python libraries and applications which address the needs of current and future work in bioinformatics. The source code is made available under the Biopython License, which is extremely liberal and compatible with almost every license in the world. We work along with the Open Bioinformatics Foundation, who generously host our website, bug tracker, and mailing lists.

This wiki will help you download and install Biopython, and start using the libraries and tools.

Get Started

- Download Biopython
- Installation help (PDF)

Get help

- Tutorial (PDF)
- Documentation on this wiki
- Cookbook (working examples)
- Join the mailing lists

Contribute

- What's being worked on
- Developing on Github

The latest release is Biopython 1.52, released on 22 September 2009.

O|BI|F News

- Working with FASTQ files in Biopython when speed matters
- Biopython CVS to git migration
- Biopython 1.52 released
- Simpler, optimized format conversion with Biopython
- Indexing sequence files with Biopython
- Biopython 1.51 released
- Biopython 1.51 beta released
- Clever tricks with NCBI Entrez EInfo (& Biopython)
- Dropping Python 2.3 Support
- Introducing (and expanding) the Biopython Cookbook

See also our news page, and twitter.
What can Biopython do?

- Connects to databases and retrieves data
- Reads (parses) various file formats → fasta, genbank, swissprot, pdb etc...
- Able to perform various sequence operations (translation, reverse complement, ...) 
- Runs Blast locally → reads the results
Biopython is huge → possibly too large?

- **Bio.Crystal**: Module to represent the NDB Atlas structure (a minimal subset of PDB format).
- **Bio.Data**: Collections of various bits of useful biological data.
  - Bio.Data.CodonTable
  - Bio.Data.IUPACData
- **Bio.EZRetrieve**: This module contains code to access EZRetrieve (DEPRECATED).
- **Bio.Emboss**: Code to interact with the ever-so-useful EMBOSS programs.
  - Bio.Emboss.Applications: Code to interact with and run various EMBOSS programs.
  - Bio.Emboss.Primer3: Code to interact with the primer3 program.
  - Bio.Emboss.PrimerSearch: Code to interact with the primerssearch program from EMBOSS.
- **Bio.Encodings**: Properties for functionality such as transcription and translation.
  - Bio.Encodings.IUPACEncoding
- **Bio.Entrez**: Provides code to access NCBI over the WWW.
- **Bio.Enzyme**: This module provides code to work with the enzyme.dat file from Enzyme (OBSOLETE as of Biopython version 1.50).
- **Bio.ExPASy**: This module provides code to access resources at ExPASy over the WWW.
  - Bio.ExPASy.Enzyme: This module provides code to work with the enzyme.dat file from Enzyme.
  - Bio.ExPASy.Prodoc: This module provides code to work with the prosite.doc file from Prosite.
  - Bio.ExPASy.Prosite: This module provides code to work with the prosite.dat file from Prosite.
  - Bio.ExPASy.ScanProsite
- **Bio.FSSP**: Parser for FSSP files, used in a database of protein fold classifications.
  - Bio.FSSP.FSSPTools
  - Bio.FSSP.fssp_rec
- **Bio.Fasta**: Utilities for working with FASTA formatted sequences (DEPRECATED).
  - Bio.Fasta.FastaAlign: Code to deal with alignments written in Fasta format (DEPRECATED).
- **Bio.GA**: A selection of genetic algorithm code.
  - Bio.GA.Crossover
    - Bio.GA.Crossover.General: General functionality for crossover that doesn't apply.
    - Bio.GA.Crossover.GenPoint: Generalized N-Point Crossover.
    - Bio.GA.Crossover.Point: Perform two-point crossovers between the genomes of two organisms.
    - Bio.GA.Crossover.TwoPoint: Perform two-point crossovers between the genomes of two organisms.
    - Bio.GA.Crossover.Uniform: Perform uniform crossover between the genomes of two
Using Entrez from Biopython

• Search for the ids or accession numbers (if not known)

  Entrez.esearch

• Fetch the data based on the accession numbers

  Entrez.efetch
Connecting to Entrez – list of resources

```python
# connecting to Entrez from biopython
from Bio import Entrez

# tell NCBI who you are
Entrez.email = "myemail@psu.edu"

# search terms
term = "Cypripedioideae[Orgn] AND matK[Gene]"

stream = Entrez.esearch(db="nucleotide", term=term)
results = Entrez.read(stream)

print results
```
The search result ➔ dictionary-like object

```python
# connecting to Entrez from biopython
from Bio import Entrez

# tell NCBI who you are
Entrez.email = "myemail@psu.edu"

# search terms
term = "Cypripedioideae[Orgn] AND matK[Gene]"

stream = Entrez.esearch(db="nucleotide", term=term)
results = Entrez.read(stream)

print results["IdList"]
```


More features of Entrez

• Entrez.esummary()  ➔ object summaries

• Entrez.elink()  ➔ finds related items

• Entrez.equery()  ➔ counts for a search term in each of the Entrez databases

• Entrez.espell()  ➔ spelling corrections
Genbank data retrieval – fasta file

```python
from Bio import Entrez

# access the resource
stream = Entrez.efetch(db="nucleotide", rettype="fasta", id="6273291")

# read the content of the stream
data = stream.read()

print data[:200]
```
Some file types have more information
Save query results to a file (responsible use of NCBI resources)
About accession numbers

• There is a unique global id (numbers): 6273291

• Within each database there is a letter+number combination that identifies the database

• For example genbank there are two letters + six digits

6273291 is called as AF191665 in Genbank
Valid parameters for Entrez.efecth()

• database = nucleotide, gene, genome, protein

• rettype = fasta, gb, gp (protein), est

• id = global id or accession number

Not all parameter combinations are valid.

If you use accession numbers they’ll have to match the database
You may search for multiple ids and save to file

```python
from Bio import Entrez

# you may search for multiple ids at the same time
ids = "6273291,6273290,6273289"

stream = Entrez.efetch(db="nucleotide", rettype="fasta", id=ids)
data = stream.read()

out = file('output.fasta', 'wt')
out.write(data)
out.close()
```
import string

# data consists of numbers
data = range(10)

# turn the numbers into strings
data = map(str, data)

# join the strings with separators
text1 = string.join(data, ', ', ')
text2 = string.join(data, '---')
text3 = string.join(data, '\t')

print text1
print text2
print text3
Sequence objects

```python
from Bio.Seq import Seq

seq = Seq('GATTACA')
mrna = seq.transcribe()
prot = seq.translate()

# this is a nucleotide sequence object
print(type(seq), seq)

# transcribe to mRNA
print(type(mrna), mrna)

# Aspartic acid and Tyrosine
print(type(prot), prot)
```

```
<class 'Bio.Seq.Seq'> GATTACA
<class 'Bio.Seq.Seq'> GAUUACA
<class 'Bio.Seq.Seq'> DY
```
Slices are also sequences

```python
from Bio.Seq import Seq

seq = Seq('GATTACA')

print(seq[:4])

print(seq[-4:])

print(seq[-4:].complement())

print(seq[-4:].reverse_complement())
```
Sequence objects ➔ to string

```python
from Bio.Seq import Seq

seq = Seq('GATTACA')

text1 = '%s' % seq
text2 = seq.tostring()

print type(seq), seq
print type(text1), text1
print type(text2), text2
```
Reading the sequences: SeqIO module

```python
from Bio import SeqIO

stream = file('output.gb')
parser = SeqIO.parse(stream, "gb")
records = list(parser)

# get the first record
first = records[0]

# a seqline object
print type(first)

# the annotation attribute is a dictionary like object
print first.annotations
```

```
<class 'Bio.SeqRecord.SeqRecord'>
{'sequence_version': 1, 'source': 'chloroplast Opuntia marenseae',
 'taxonomy': ['Eukaryota', 'Viridiplantae', 'Streptophyta', 'Embryophyta',
 'Tracheophyta', 'Spermatophyta', 'Magnoliophyta', 'eudicotyledons', 'core
eudicotyledons', 'Caryophyllales', 'Cactaceae', 'Opuntioideae', 'Opuntia'],
'keywords': [''], 'references': [Reference(title='Phylogeny of the subfamily Opuntioideae (Cactaceae)', ...), Reference(title='Direct Submission', ...)], 'accessions': ['AF191665'], 'data_file_division':
'PLN', 'date': '07-NOV-1999', 'organism': 'Opuntia marenseae', 'gi': '6273291'}
```
SeqRecord objects

```python
from Bio import SeqIO

stream = file('output.gb')
parser = SeqIO.parse(stream, "gb")
records = list(parser)

# get the first record
first = records[0]
print type(first)
print type(first.name), first.id
print type(first.seq), first.seq
```
Long story short

- `SeqIO.parse` returns an iterable where each element is an object of `SeqRecord` type.

  Parse → [ SeqRecord, SeqRecord, SeqRecord ]

- Each `SeqRecord` object has many attributes, one of them is called `seq` and that contains the sequence as an object of `Seq` type:

  SeqRecord → record.seq → Seq

  SeqRecord → record.annotations → dictionary
SeqRecord usage example

```python
from Bio import SeqIO

stream = file('output.gb')
parser = SeqIO.parse(stream, "gb")
records = list(parser)

for rec in records:
    size = len(rec.seq)
    print rec.id, size, rec.seq[:10]
```
General tips

• Identify accession numbers first – either from python or via web browser at NCBI

• Fetch the results and save them into a file

• Parse the result files to extract the information
Homework

• Retrieve ids 6273291, 6273290, 6273289 from Entrez and save them into a file in genbank format

• Parse this file and print the answer to the following questions:

  1. what organism corresponds to each id
  2. when was the data submitted to Entrez
  3. what are the last 10 nucleotides for each sequence
Week 12 - Lecture 24

István Albert
Huck Institutes for the Life Sciences
PCR amplification

- **Denaturation:** 95°C
- **Annealing:** ~50°C
- **Extension:** 72°C

(A)
Today’s topic → based on a submitted project

• Given a number of GenBank ids we want to decide whether the sequences corresponding to these ids can have unique (forward, reverse) primers designed for them

We aren’t going to choose the primers ourselves there are some other rules need to be followed

We just want to find out whether or not there are unique combinations that we could choose from
Let’s get the sequence ids from the file.

```python
import bmm

rows = bmm.read_tabular('ecoli-flic-gene-ids.txt')
ids = bmm.string_column(rows, 'GenbankID')

print len(ids)
print ids
```

Command Output:
```
52
['AB028471', 'AY249138', 'AB128919*', 'AY249989', 'AB028473', 'AY249991', 'AB028474', 'AY249993', 'AY249994', 'AY249995', 'AY249996', 'AB028475', 'AY249998', 'AY249999', 'AY250000', 'AJ515904', 'AY250001', 'AY250002', 'AY250003', 'AY250004']
```
Clean up the input – valid genbank ids

```
6  def strip(name):
7    return name.strip('*)
8
9  ids = map(strip, ids)
10
11  print len(ids)
12  print ids
```

Command Output

```
52
['AB028471', 'AY249138', 'AB128919', 'AY249989',
 'AB028473', 'AY249991', 'AB028474', 'AY249993',
 'AY249994', 'AY249995', 'AY249996', 'AB028475',
 'AY249998', 'AY249999', 'AY250000', 'AJ515904',
 'AY250001', 'AY250002', 'AY250003', 'AY250004']
```
Fetch sequences from GenBank – save to a file

```python
out = file('ecoli.gb', 'wt')
for id in ids:
    print 'Fetching id', id
    stream = Entrez.efetch(db='nucleotide', rettype='gb', id=id)
    data = stream.read()
    out.write(data)
out.close()
```

Also needs: `from Bio import Entrez` at the top
Inspect what you have – with a new program

```python
# this is a new program

from Bio import SeqIO

stream = file('ecoli.gb')

for record in SeqIO.parse(stream, 'gb'):
    print record.id, len(record.seq)
```
First task

• If we were to look at the first five bases, how many unique starts can we find across all 52 sequences?
Extract the first 5 bases

```python
from Bio import SeqIO

stream = file('ecoli.gb')
records = list(SeqIO.parse(stream, 'gb'))

# extract first 5 bases as a string

def extract(record):
    seq = record.seq[:5]
    return seq.tostring()

data = map(extract, records)

print data[:3]
```
Unique elements $\Rightarrow$ set()

def extract(record):
    seq = record.seq[:5]
    return seq.tostring()

data = map(extract, records)

# to check for uniqueness compare length of the list to the length of the set
uniq = set(data)
print "Size=%s, total=%s, unique=%s" % (5, len(data), len(uniq))
Homework 1/3

• Expand the previous program to test for increasing slices.

• What is the smallest slice that guarantees uniqueness over all sequences?

• Print out the size of the slice, the number of total elements and number of unique sequences (see next slide)
```python
stream = file('ecoli.gb')
records = list(SeqIO.parse(stream, 'gb'))

print "Size=%s, total=%s, unique=%s" % (size, len(data), len(uniq))
```
Homework 2/3

- Answer the same questions as Homework 1 but for the end N slices

- The beauty of functional programming is that all you need to change is one line (swap 2 words)
• Primers uniquely identify a region if the (forward, reverse) pair is unique → thus these are unique pairs:

('AAA', 'TAT')
('AAA', 'TAA')
('AAT', 'TAA')
('AAT', 'TAT')
Small detour: only immutable objects may be stored in sets

Good news: primer pairs tuples can be stored in sets – the set will ensure their uniqueness
Only (start, end) pairs need to be unique

```python
from Bio import SeqIO

stream = file('ecoli.gb')
records = list(SeqIO.parse(stream, 'gb'))

def extract(record):
    fwd = record.seq[:5].tostring()
    rev = record.seq[-5:].tostring()

    return fwd, rev

data = map(extract, records)

print data[:3]
```

```
[('GCGAT', 'ACTCA'), ('AACAA', 'CGCAG'), ('ATTAC', 'CTTAT')]
```
Homework 3/3

• Answer the same questions as Homework 1 but for the uniqueness of (start, end) pairs for increasing slices.

• How large do the slices need to be to have unique pairings?

Tip:
Create three extractor functions.
That way you can map any one of them and you will be able to recreate each answer at any time
Where to go from here – for the project

- It may be that there are two very similar sequences that cause the primers to be too long, thus if one of them were removed the size of the primers could be greatly reduced.

- **Task:** for each primer pair find which sequences contain that pairing.

Optional *(bonus)* homework
We need to create a primer \(\rightarrow\) id dictionary mapping where keys are primer pairs and values are lists of ids.
Create a dictionary with list values

```python
store = {}
for pair in primers:
    store[pair] = []

# what does the collated structure contain?
collated = zip(primers, ids)

# this here does all the work
for pair, id in collated:
    store[pair].append(id)

# printing the results
for key, value in store.items():
    print(key, len(value), value)
```

```python
('ACTCGTCTCT', 'TCTCCCCCGA') 1 ['EF392693.1']
('ATGGCACAAG', 'GCAGGGTTAA') 25 ['AY249991.1', 'AY249994.1', 'AY249995.1', 'AY249998.1', 'AY250001.1', 'AY250002.1', 'AY250003.1', 'AY250008.1', 'AY250010.1', 'AY250012.1', 'AY250011.1', 'AY250013.1', 'AY250014.1', 'AY250015.1', 'AY250017.1', 'AY250018.1', 'AY250020.1', 'AY250021.1', 'AY250022.1', 'AY250023.1', 'AY250024.1', 'AY250025.1', 'AY250026.1']
```
Bonus Homework

• Find out what sequences cause the most non-uniqueness to occur in the 30 to 60 bp slice range

Print out only if there is more than 1 match