Biopython Tutorial and Cookbook

Je Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock

Last Update–16 March 2007

Contents

Chapter 2

Quick Start – What can you do with Biopython?

```
>>> new_seq = my_seq[0:5]
>>> print new_seq
Seq('GATCG', IUPACUnambiguousDNA())
>>> my_seq + new_seq
Seq('GATCGATGGGCCTATATAGGATCGAAAATCGCGATCG', IUPACUnambiguousDNA())
>>> my_seq[5]
'A'
>>> my_seq == new_seq
True
```

In all of the operations, the alphabet property is maintained. This is very useful in case you accidentally end up trying to do something weird like add a protein sequence and a DNA sequence:

```
>>> protein_seq = Seq('EVRNAK', IUPAC.protein)
>>> dna_seq = Seq('ACGT', IUPAC.unambiguous_dna)
>>> protein_seq + dna_seq
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/usr/local/liblocal/<thon1.6/site-pck line          42cal in ?__add__]TJ10.461-11.955Td[(Frais)
```

```
>>> from Bio import Transcribe
>>> transcriber = Transcribe.unambiguous_transcriber
>>> my_rna_seq = transcriber.transcribe(my_seq)
>>> print my_rna_seq
Seq('GAUCGAUGGGCCUAUAUAGGAUCGAAAAUCGC', IUPACUnambiguousRNA())
```

The alphabet of the new RNA Seq object is created for free, so again, dealing with a Seq object is no more difficult then dealing with a simple string.

You can also reverse transcribe RNA sequences:

```
>>> transcriber.back_transcribe(my_rna_seq)
Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC', IUPACUnambiguousDNA())
```

```
from Bio import SeqIO
handle = open("ls_orchid.fasta")
```

Then we can give this function to the `SeqIO.to_dict` function to use in building the dictionary:

```
from Bio import SeqIO
handle = open("ls_orchid.fasta")
orchid_dict = SeqIO.to_dict(SeqIO.parse(handle, "fasta"), key_function=get_accession)
handle.close()
print orchid_dict.keys()
```

Finally, as desired, the new dictionary keys:

```
>>> print orchid_dict.keys()
['Z78484.1', 'Z78464.1', 'Z78455.1', 'Z78442.1', 'Z78532.1', 'Z78453.1', ..., 'Z78471.1']
```

Not too complicated, I hope!

2.4.6 Extracting data

2.5 Connecting with biological databases

One of the very common things that you need to do in bioinformatics is extract information from biological databases. It can be quite tedious to access these databases manually, especially if you have a lot of repetitive work to do. Biopython attempts to save you time and energy by making some on-line databases available from python scripts. Currently, Biopython has code to extract information from the following databases:

- ExPASy – See section 4.1 in the Cookbook for more information.
- Entrez from NCBI – See below
- PubMed from NCBI

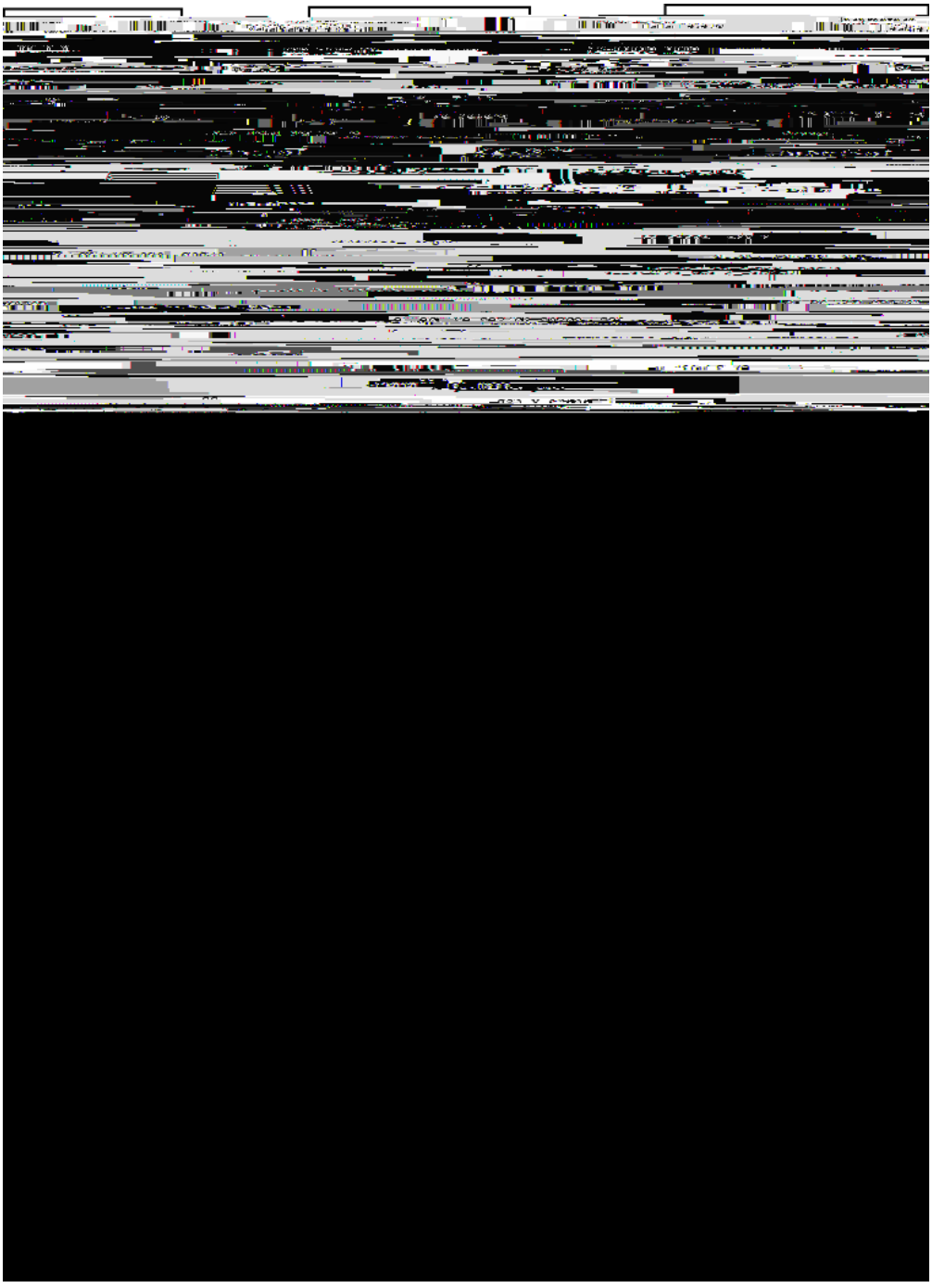
```
result_file = open(result_file_name, "w")
result_file.write(result_handle.read())
result_file.close()

if my_browser == "lynx":
    os.system("lynx -force_html " + result_file_name)
elif my_browser == "netscape":
    os.system("netscape file:" + result_file_name)
```

Chapter 3

BLAST

The qblast function also take a number of other option arguments which are basically analogous to the



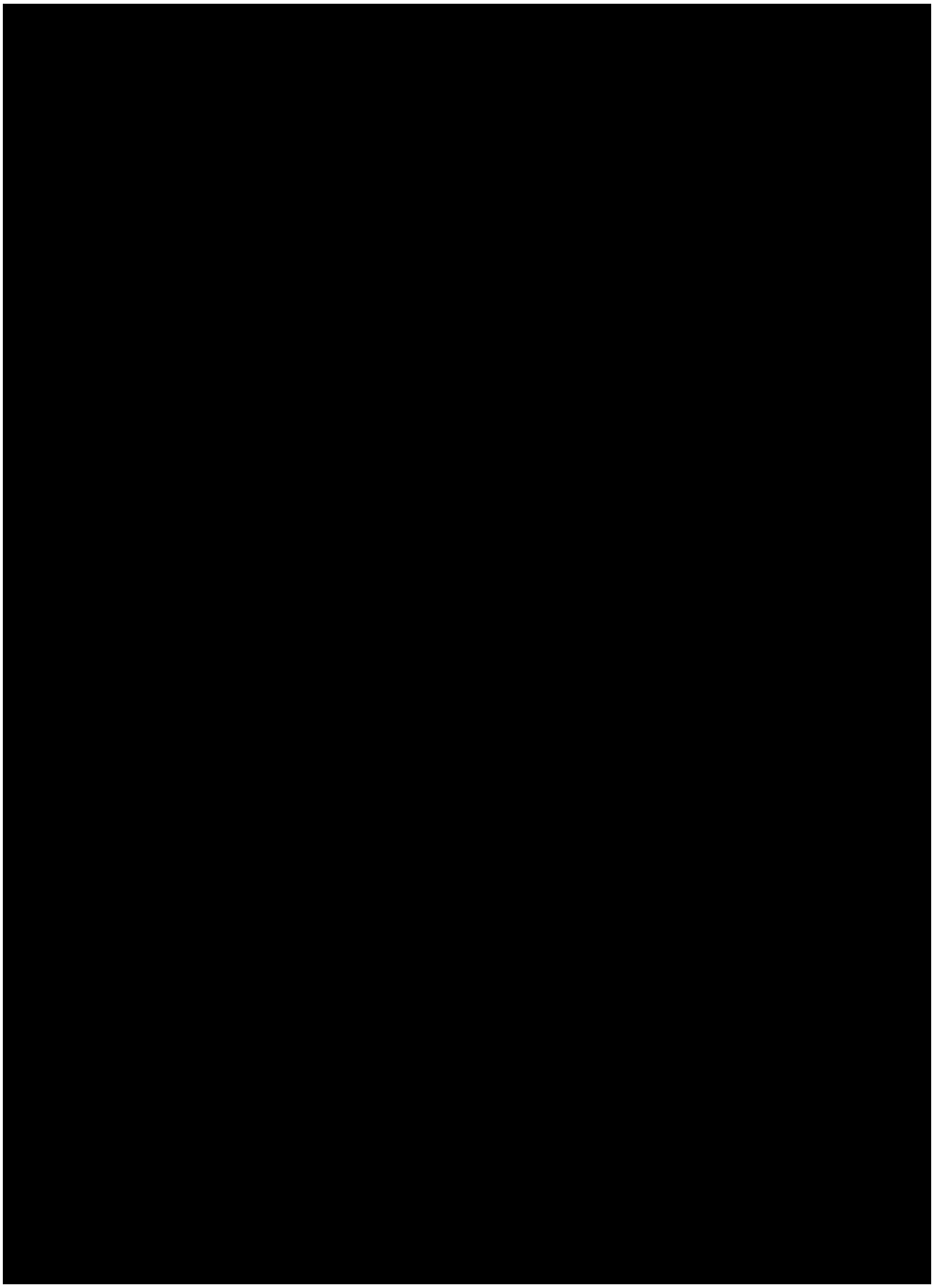


Figure 3.2: Class diagram for the PSIBlast Record class.

```
...         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] + '...'
```

If you also read the section 3.4 on parsing BLAST XML output, you'll notice that the above code is identical to what is found in that section. Once you parse something into a record class you can deal with

Right now the `BlastErrorParser`

```
# handle = ExPASy.sprot_search_ful("Orchid and {Chalcone Synthase}")
html_results = handle.read()
if "Number of sequences found" in html_results:
```

The output for this looks like:

```
>>> from Bio import GenBank
>>> dict_file = 'cor6_6.gb'
>>> index_file = 'cor6_6.idx'
>>> GenBank.index_file(dict_file, index_file)
```

This will create a directory called `cor6_6.idx` containing the index files. Now, we can use this index to

4.4.2 Calculating summary information

```
GTATC
AT--C
CTGTC
```

the PSSM for this alignment is:

```
G A T C
```

4.4.5 Information Content

type – This is a textual description of the type of feature (for instance, this will be something like 'CDS' or 'gene').

ref

position

We can access the fuzzy start and end positions using the start and end attributes of the location:

```
>>> my_location.start
<Bio.SeqFeature.AfterPosition instance at 0x101d7164>
>>> print my_location.start
>5
```

38,c-5625Lastc-5625annotationc-5625update)

4.10.1.1 Structure

The second field in the Residue id is the sequence identifier, an integer describing the position of the residue in the chain.

The third field is a string, consisting of the insertion code. The insertion code is sometimes used to preserve a certain desirable residue numbering scheme. A Ser 80 insertion mutant (inserted e.g. between a Thr 80 and an Asn 81 residue) could e.g. have sequence identifiers and insertion codes as followed: Thr 80

```
for residue in chain.get_list():
residue_id=residue.get_id()
hetfield=residue_id[0]
if hetfield[0]=="H":
print residue_id
```

Print out the coordinates of all CA atoms in a structure with B factor greater than 50.

```
for model in structure.get_list():
  for chain in model.get_list():
    for residue in chain.get_list():
      if residue.has_id("CA"):
        ca=residue["CA"]
        if ca.get_bfactor()>50.0:
          print ca.get_coord()
```

Print out all the residues that contain disordered atoms.

```
for model in structure.get_list()
  for chain in model.get_list():
    for residue in chain.get_list():
      if residue.is_disordered():
        resseq=residue.get_id()[1]
        resname=residue.get_resname()
        model_id=model.get_id()
        chain_id=chain.get_id()
        print model_id, chain_id, resname, resseq
```

Loop over all disordered atoms, and select all atoms with altloc A (if present). This will make sure that the SMCRA data structure will behave as if only the atoms with altloc A are present.

4.10.5.1.1 Duplicate residues One structure contains two amino acid residues in one chain with the same sequence identifier (resseq 3) and icode. Upon inspection it was found that this chain contains the

4.10.6 Othta0G15Tf7st015TfeaturTf7ss

Chapter 5

Advanced

database
posted_date
num_letters_in_database
num_sequences_in_database
num_letters_searched RESERVED. Currently unused. I’ve never

5.3.8 KEGG

5.3.8.1 Bio.KEGG.Enzyme

The Bio.KEGG.Enzyme module works with the 'enzyme' file from the Ligand database, which can be obtained from the KEGG project. (http://www.genome.ad.jp/kegg).

The Bio.KEGG.Enzyme.Record contains all the information stored in a KEGG/Enzyme record. Its string representation also is a valid KEGG record, but it is NOT guaranteed to be equivalent to the record from which it was produced.

The Bio.KEGG.Enzyme.Scanner produces the following events:

```
entry
name
classname
sysname
reaction
substrate
product
inhibitor
cofactor
effector
comment
pathway_db
pathway_id
pathway_desc
organism
gene_id
```

```
enzyme_role
structure_db
structure_id
dblinks_db
dblinks_id
record_end
```

5.3.9 Fasta

record_originator
journal_subset
subheadings
secondary_source_id
source
title_abbreviation
title
transliterated_title
unique_identifier
volume_issue
year
pubmed_id

organelle
organism_classification
reference_number
reference_position
reference_comment
reference_cross_reference
reference_author
reference_title
reference_location
comment
database_cross_reference
keyword
featu3Fabitledaer

5.3.15 MetaTool

The MetaTool parser works with MetaTool output files. MetaTool implements algorithms to decompose a

The MetaTool (a)]Teb page is

(d) `self.sum_letters`: a dictionary. `{i1: s1, i2: s2,...,in:sn}` where:

i.

Chapter 6

Where to go from here – contributing to Biopython

6.1 Maintaining a distribution for a platform

We try to release Biopython to make it as easy to install as possible for users. Thus, we try to provide the Biopython libraries in as many install formats as we can. Doing this from release to release can be a lot of work for developers, and sometimes requires them to maintain packages they are not all that familiar with. This section is meant to provide tipsTf0-a27(e)-347(c)-1(an)1(.)-486(Doir)-3-1(an)o955Td[0(Th)(o)27ns,a3347(l)1(me)-1e

Macintosh – We would love to find someone who wants to maintain a Macintosh distribution, and make it available in a Macintosh friendly format like bin-hex. This would basically include finding a way to compile everything on the Mac, making sure all of the code written by us UNIX-based developers works well on the Mac, and providing any Mac-friendly hints for us.

Once you've got a package, please test it on your system to make sure it installs everything in a good way and seems to work properlyv

Chapter 7

Appendix: Useful stu about Python

If you haven't spent a lot of time programming in python, many questions and problems that come up in

```
 with multiple lines.
>>> import cStringIO
>>> my_info_handle = cStringIO.StringIO(my_info)
>>> first_line = my_info_handle.readline()
>>> print first_line
A string

>>> second_line = my_info_handle.readline()
>>> print second_line
 with multiple lines.
```