Biopython Tutorial and Cookbook

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

Last Update – 16 December 2015 (Biopython 1.66+)

Contents

1 Introduction **9**
- 1.1 What is Biopython? 9
- 1.2 What can I find in the Biopython package 9
- 1.3 Installing Biopython 10
- 1.4 Frequently Asked Questions (FAQ) 11

2 Quick Start – What can you do with Biopython? **15**
- 2.1 General overview of what Biopython provides 15
- 2.2 Working with sequences 15
- 2.3 A usage example 16
- 2.4 Parsing sequence file formats 17
  - 2.4.1 Simple FASTA parsing example 17
  - 2.4.2 Simple GenBank parsing example 18
  - 2.4.3 I love parsing – please don't stop talking about it! 18
- 2.5 Connecting with biological databases 18
- 2.6 What to do next 19

3 Sequence objects **20**
- 3.1 Sequences and Alphabets 20
- 3.2 Sequences act like strings 21
- 3.3 Slicing a sequence 22
- 3.4 Turning Seq objects into strings 23
- 3.5 Concatenating or adding sequences 23
- 3.6 Changing case 25
- 3.7 Nucleotide sequences and (reverse) complements 25
- 3.8 Transcription 26
- 3.9 Translation 27
- 3.10 Translation Tables 29
- 3.11 Comparing Seq objects 30
- 3.12 MutableSeq objects 31
- 3.13 UnknownSeq objects 33
- 3.14 Working with strings directly 34

4 Sequence annotation objects **35**
- 4.1 The SeqRecord object 35
- 4.2 Creating a SeqRecord 36
  - 4.2.1 SeqRecord objects from scratch 36
  - 4.2.2 SeqRecord objects from FASTA files 37
  - 4.2.3 SeqRecord objects from GenBank files 38
- 4.3 Feature, location and position objects 39

4.3.1 SeqFeature objects . . . 39
4.3.2 Positions and locations . . . 40
4.3.3 Sequence described by a feature or location . . . 43
4.4 Comparison . . . 44
4.5 References . . . 44
4.6 The format method . . . 45
4.7 Slicing a SeqRecord . . . 45
4.8 Adding SeqRecord objects . . . 47
4.9 Reverse-complementing SeqRecord objects . . . 49

5 Sequence Input/Output 51
5.1 Parsing or Reading Sequences . . . 51
5.1.1 Reading Sequence Files . . . 51
5.1.2 Iterating over the records in a sequence file . . . 52
5.1.3 Getting a list of the records in a sequence file . . . 53
5.1.4 Extracting data . . . 54
5.2 Parsing sequences from compressed files . . . 56
5.3 Parsing sequences from the net . . . 57
5.3.1 Parsing GenBank records from the net . . . 57
5.3.2 Parsing SwissProt sequences from the net . . . 58
5.4 Sequence files as Dictionaries . . . 58
5.4.1 Sequence files as Dictionaries – In memory . . . 59
5.4.2 Sequence files as Dictionaries – Indexed files . . . 61
5.4.3 Sequence files as Dictionaries – Database indexed files . . . 63
5.4.4 Indexing compressed files . . . 63
5.4.5 Discussion . . . 64
5.5 Writing Sequence Files . . . 65
5.5.1 Round trips . . . 66
5.5.2 Converting between sequence file formats . . . 67
5.5.3 Converting a file of sequences to their reverse complements . . . 67
5.5.4 Getting your SeqRecord objects as formatted strings . . . 68

6 Multiple Sequence Alignment objects 70
6.1 Parsing or Reading Sequence Alignments . . . 70
6.1.1 Single Alignments . . . 71
6.1.2 Multiple Alignments . . . 73
6.1.3 Ambiguous Alignments . . . 75
6.2 Writing Alignments . . . 77
6.2.1 Converting between sequence alignment file formats . . . 78
6.2.2 Getting your alignment objects as formatted strings . . . 80
6.3 Manipulating Alignments . . . 81
6.3.1 Slicing alignments . . . 81
6.3.2 Alignments as arrays . . . 84
6.4 Alignment Tools . . . 84
6.4.1 ClustalW . . . 85
6.4.2 MUSCLE . . . 86
6.4.3 MUSCLE using stdout . . . 87
6.4.4 MUSCLE using stdin and stdout . . . 88
6.4.5 EMBOSS needle and water . . . 90

7 BLAST **92**
7.1 Running BLAST over the Internet . . . 92
7.2 Running BLAST locally . . . 94
7.2.1 Introduction . . . 94
7.2.2 Standalone NCBI BLAST+ . . . 94
7.2.3 Other versions of BLAST . . . 95
7.3 Parsing BLAST output . . . 95
7.4 The BLAST record class . . . 97
7.5 Deprecated BLAST parsers . . . 98
7.5.1 Parsing plain-text BLAST output . . . 98
7.5.2 Parsing a plain-text BLAST file full of BLAST runs . . . 101
7.5.3 Finding a bad record somewhere in a huge plain-text BLAST file . . . 102
7.6 Dealing with PSI-BLAST . . . 103
7.7 Dealing with RPS-BLAST . . . 103

8 BLAST and other sequence search tools (*experimental code*) **104**
8.1 The SearchIO object model . . . 105
8.1.1 QueryResult . . . 105
8.1.2 Hit . . . 110
8.1.3 HSP . . . 113
8.1.4 HSPFragment . . . 116
8.2 A note about standards and conventions . . . 117
8.3 Reading search output files . . . 118
8.4 Dealing with large search output files with indexing . . . 118
8.5 Writing and converting search output files . . . 119

9 Accessing NCBI's Entrez databases **121**
9.1 Entrez Guidelines . . . 122
9.2 EInfo: Obtaining information about the Entrez databases . . . 123
9.3 ESearch: Searching the Entrez databases . . . 125
9.4 EPost: Uploading a list of identifiers . . . 125
9.5 ESummary: Retrieving summaries from primary IDs . . . 126
9.6 EFetch: Downloading full records from Entrez . . . 126
9.7 ELink: Searching for related items in NCBI Entrez . . . 129
9.8 EGQuery: Global Query - counts for search terms . . . 131
9.9 ESpell: Obtaining spelling suggestions . . . 131
9.10 Parsing huge Entrez XML files . . . 131
9.11 Handling errors . . . 132
9.12 Specialized parsers . . . 134
9.12.1 Parsing Medline records . . . 135
9.12.2 Parsing GEO records . . . 137
9.12.3 Parsing UniGene records . . . 137
9.13 Using a proxy . . . 139
9.14 Examples . . . 139
9.14.1 PubMed and Medline . . . 139
9.14.2 Searching, downloading, and parsing Entrez Nucleotide records . . . 140
9.14.3 Searching, downloading, and parsing GenBank records . . . 142
9.14.4 Finding the lineage of an organism . . . 143
9.15 Using the history and WebEnv . . . 144
9.15.1 Searching for and downloading sequences using the history . . . 144
9.15.2 Searching for and downloading abstracts using the history . . . 146
9.15.3 Searching for citations . . . 146

10 Swiss-Prot and ExPASy **148**
10.1 Parsing Swiss-Prot files . . . 148
10.1.1 Parsing Swiss-Prot records . . . 148
10.1.2 Parsing the Swiss-Prot keyword and category list . . . 150
10.2 Parsing Prosite records . . . 151
10.3 Parsing Prosite documentation records . . . 152
10.4 Parsing Enzyme records . . . 152
10.5 Accessing the ExPASy server . . . 154
10.5.1 Retrieving a Swiss-Prot record . . . 154
10.5.2 Searching Swiss-Prot . . . 155
10.5.3 Retrieving Prosite and Prosite documentation records . . . 155
10.6 Scanning the Prosite database . . . 156

11 Going 3D: The PDB module **158**
11.1 Reading and writing crystal structure files . . . 158
11.1.1 Reading a PDB file . . . 158
11.1.2 Reading an mmCIF file . . . 159
11.1.3 Reading files in the PDB XML format . . . 159
11.1.4 Writing PDB files . . . 159
11.2 Structure representation . . . 160
11.2.1 Structure . . . 162
11.2.2 Model . . . 163
11.2.3 Chain . . . 163
11.2.4 Residue . . . 163
11.2.5 Atom . . . 164
11.2.6 Extracting a specific `Atom/Residue/Chain/Model` from a Structure . . . 165
11.3 Disorder . . . 166
11.3.1 General approach . . . 166
11.3.2 Disordered atoms . . . 166
11.3.3 Disordered residues . . . 166
11.4 Hetero residues . . . 167
11.4.1 Associated problems . . . 167
11.4.2 Water residues . . . 167
11.4.3 Other hetero residues . . . 167
11.5 Navigating through a Structure object . . . 167
11.6 Analyzing structures . . . 170
11.6.1 Measuring distances . . . 170
11.6.2 Measuring angles . . . 170
11.6.3 Measuring torsion angles . . . 170
11.6.4 Determining atom-atom contacts . . . 171
11.6.5 Superimposing two structures . . . 171
11.6.6 Mapping the residues of two related structures onto each other . . . 171
11.6.7 Calculating the Half Sphere Exposure . . . 171
11.6.8 Determining the secondary structure . . . 172
11.6.9 Calculating the residue depth . . . 172
11.7 Common problems in PDB files . . . 173
11.7.1 Examples . . . 173
11.7.2 Automatic correction . . . 174
11.7.3 Fatal errors . . . 174
11.8 Accessing the Protein Data Bank . . . 175
11.8.1 Downloading structures from the Protein Data Bank . . . 175
11.8.2 Downloading the entire PDB . . . 175

11.8.3 Keeping a local copy of the PDB up to date . . . 175
11.9 General questions . . . 176
11.9.1 How well tested is Bio.PDB? . . . 176
11.9.2 How fast is it? . . . 176
11.9.3 Is there support for molecular graphics? . . . 176
11.9.4 Who's using Bio.PDB? . . . 176

12 Bio.PopGen: Population genetics 177
12.1 GenePop . . . 177
12.2 Coalescent simulation . . . 179
12.2.1 Creating scenarios . . . 179
12.2.2 Running Fastsimcoal2 . . . 181
12.3 Other applications . . . 182
12.3.1 FDist: Detecting selection and molecular adaptation . . . 182
12.4 Future Developments . . . 185

13 Phylogenetics with Bio.Phylo 186
13.1 Demo: What's in a Tree? . . . 186
13.1.1 Coloring branches within a tree . . . 187
13.2 I/O functions . . . 190
13.3 View and export trees . . . 191
13.4 Using Tree and Clade objects . . . 195
13.4.1 Search and traversal methods . . . 195
13.4.2 Information methods . . . 197
13.4.3 Modification methods . . . 197
13.4.4 Features of PhyloXML trees . . . 198
13.5 Running external applications . . . 198
13.6 PAML integration . . . 199
13.7 Future plans . . . 199

14 Sequence motif analysis using Bio.motifs 201
14.1 Motif objects . . . 201
14.1.1 Creating a motif from instances . . . 201
14.1.2 Creating a sequence logo . . . 203
14.2 Reading motifs . . . 204
14.2.1 JASPAR . . . 204
14.2.2 MEME . . . 210
14.2.3 TRANSFAC . . . 213
14.3 Writing motifs . . . 216
14.4 Position-Weight Matrices . . . 217
14.5 Position-Specific Scoring Matrices . . . 218
14.6 Searching for instances . . . 219
14.6.1 Searching for exact matches . . . 219
14.6.2 Searching for matches using the PSSM score . . . 220
14.6.3 Selecting a score threshold . . . 220
14.7 Each motif object has an associated Position-Specific Scoring Matrix . . . 221
14.8 Comparing motifs . . . 224
14.9 *De novo* motif finding . . . 225
14.9.1 MEME . . . 225
14.9.2 AlignAce . . . 226
14.10Useful links . . . 227

15 Cluster analysis **228**
15.1 Distance functions 229
15.2 Calculating cluster properties 232
15.3 Partitioning algorithms 234
15.4 Hierarchical clustering 237
15.5 Self-Organizing Maps 241
15.6 Principal Component Analysis 243
15.7 Handling Cluster/TreeView-type files 244
15.8 Example calculation 249
15.9 Auxiliary functions 249

16 Supervised learning methods **250**
16.1 The Logistic Regression Model 250
16.1.1 Background and Purpose 250
16.1.2 Training the logistic regression model 251
16.1.3 Using the logistic regression model for classification 253
16.1.4 Logistic Regression, Linear Discriminant Analysis, and Support Vector Machines . . . 255
16.2 *k*-Nearest Neighbors 255
16.2.1 Background and purpose 255
16.2.2 Initializing a *k*-nearest neighbors model 256
16.2.3 Using a *k*-nearest neighbors model for classification 256
16.3 Naïve Bayes 258
16.4 Maximum Entropy 258
16.5 Markov Models 258

17 Graphics including GenomeDiagram **259**
17.1 GenomeDiagram 259
17.1.1 Introduction 259
17.1.2 Diagrams, tracks, feature-sets and features 259
17.1.3 A top down example 260
17.1.4 A bottom up example 262
17.1.5 Features without a SeqFeature 262
17.1.6 Feature captions 263
17.1.7 Feature sigils 265
17.1.8 Arrow sigils 265
17.1.9 A nice example 269
17.1.10 Multiple tracks 270
17.1.11 Cross-Links between tracks 273
17.1.12 Further options 277
17.1.13 Converting old code 278
17.2 Chromosomes 278
17.2.1 Simple Chromosomes 278
17.2.2 Annotated Chromosomes 281

18 KEGG **283**
18.1 Parsing KEGG records 283
18.2 Querying the KEGG API 283

19 Cookbook – Cool things to do with it **286**
19.1 Working with sequence files . . . 286
19.1.1 Filtering a sequence file . . . 286
19.1.2 Producing randomised genomes . . . 287
19.1.3 Translating a FASTA file of CDS entries . . . 288
19.1.4 Making the sequences in a FASTA file upper case . . . 289
19.1.5 Sorting a sequence file . . . 289
19.1.6 Simple quality filtering for FASTQ files . . . 290
19.1.7 Trimming off primer sequences . . . 291
19.1.8 Trimming off adaptor sequences . . . 292
19.1.9 Converting FASTQ files . . . 293
19.1.10 Converting FASTA and QUAL files into FASTQ files . . . 295
19.1.11 Indexing a FASTQ file . . . 295
19.1.12 Converting SFF files . . . 296
19.1.13 Identifying open reading frames . . . 297
19.2 Sequence parsing plus simple plots . . . 299
19.2.1 Histogram of sequence lengths . . . 299
19.2.2 Plot of sequence GC% . . . 300
19.2.3 Nucleotide dot plots . . . 301
19.2.4 Plotting the quality scores of sequencing read data . . . 303
19.3 Dealing with alignments . . . 304
19.3.1 Calculating summary information . . . 305
19.3.2 Calculating a quick consensus sequence . . . 305
19.3.3 Position Specific Score Matrices . . . 306
19.3.4 Information Content . . . 307
19.4 Substitution Matrices . . . 309
19.4.1 Using common substitution matrices . . . 309
19.4.2 Creating your own substitution matrix from an alignment . . . 309
19.5 BioSQL – storing sequences in a relational database . . . 310

20 The Biopython testing framework **311**
20.1 Running the tests . . . 311
20.1.1 Running the tests using Tox . . . 312
20.2 Writing tests . . . 312
20.2.1 Writing a print-and-compare test . . . 313
20.2.2 Writing a unittest-based test . . . 314
20.3 Writing doctests . . . 317

21 Advanced **319**
21.1 Parser Design . . . 319
21.2 Substitution Matrices . . . 319
21.2.1 SubsMat . . . 319
21.2.2 FreqTable . . . 322

22 Where to go from here – contributing to Biopython **324**
22.1 Bug Reports + Feature Requests . . . 324
22.2 Mailing lists and helping newcomers . . . 324
22.3 Contributing Documentation . . . 324
22.4 Contributing cookbook examples . . . 324
22.5 Maintaining a distribution for a platform . . . 325
22.6 Contributing Unit Tests . . . 325
22.7 Contributing Code . . . 326

23 Appendix: Useful stuff about Python **327**
23.1 What the heck is a handle? 327
23.1.1 Creating a handle from a string 328

Chapter 1

Introduction

1.1 What is Biopython?

The Biopython Project is an international association of developers of freely available Python (http://www.python.org) tools for computational molecular biology. Python is an object oriented, interpreted, flexible language that is becoming increasingly popular for scientific computing. Python is easy to learn, has a very clear syntax and can easily be extended with modules written in C, C++ or FORTRAN.

The Biopython web site (http://www.biopython.org) provides an online resource for modules, scripts, and web links for developers of Python-based software for bioinformatics use and research. Basically, the goal of Biopython is to make it as easy as possible to use Python for bioinformatics by creating high-quality, reusable modules and classes. Biopython features include parsers for various Bioinformatics file formats (BLAST, Clustalw, FASTA, Genbank,...), access to online services (NCBI, Expasy,...), interfaces to common and not-so-common programs (Clustalw, DSSP, MSMS...), a standard sequence class, various clustering modules, a KD tree data structure etc. and even documentation.

Basically, we just like to program in Python and want to make it as easy as possible to use Python for bioinformatics by creating high-quality, reusable modules and scripts.

1.2 What can I find in the Biopython package

The main Biopython releases have lots of functionality, including:

- The ability to parse bioinformatics files into Python utilizable data structures, including support for the following formats:
  - Blast output – both from standalone and WWW Blast
  - Clustalw
  - FASTA
  - GenBank
  - PubMed and Medline
  - ExPASy files, like Enzyme and Prosite
  - SCOP, including 'dom' and 'lin' files
  - UniGene
  - SwissProt
- Files in the supported formats can be iterated over record by record or indexed and accessed via a Dictionary interface.

- Code to deal with popular on-line bioinformatics destinations such as:
  - NCBI – Blast, Entrez and PubMed services
  - ExPASy – Swiss-Prot and Prosite entries, as well as Prosite searches
- Interfaces to common bioinformatics programs such as:
  - Standalone Blast from NCBI
  - Clustalw alignment program
  - EMBOSS command line tools
- A standard sequence class that deals with sequences, ids on sequences, and sequence features.
- Tools for performing common operations on sequences, such as translation, transcription and weight calculations.
- Code to perform classification of data using k Nearest Neighbors, Naive Bayes or Support Vector Machines.
- Code for dealing with alignments, including a standard way to create and deal with substitution matrices.
- Code making it easy to split up parallelizable tasks into separate processes.
- GUI-based programs to do basic sequence manipulations, translations, BLASTing, etc.
- Extensive documentation and help with using the modules, including this file, on-line wiki documentation, the web site, and the mailing list.
- Integration with BioSQL, a sequence database schema also supported by the BioPerl and BioJava projects.

We hope this gives you plenty of reasons to download and start using Biopython!

1.3 Installing Biopython

All of the installation information for Biopython was separated from this document to make it easier to keep updated.

The short version is go to our downloads page (`http://biopython.org/wiki/Download`), download and install the listed dependencies, then download and install Biopython. Biopython runs on many platforms (Windows, Mac, and on the various flavors of Linux and Unix). For Windows we provide pre-compiled click-and-run installers, while for Unix and other operating systems you must install from source as described in the included README file. This is usually as simple as the standard commands:

```
python setup.py build
python setup.py test
sudo python setup.py install
```

(You can in fact skip the build and test, and go straight to the install – but its better to make sure everything seems to be working.)

The longer version of our installation instructions covers installation of Python, Biopython dependencies and Biopython itself. It is available in PDF (`http://biopython.org/DIST/docs/install/Installation.pdf`) and HTML formats (`http://biopython.org/DIST/docs/install/Installation.html`).

1.4 Frequently Asked Questions (FAQ)

1. *How do I cite Biopython in a scientific publication?*
   Please cite our application note [1, Cock *et al.*, 2009] as the main Biopython reference. In addition, please cite any publications from the following list if appropriate, in particular as a reference for specific modules within Biopython (more information can be found on our website):

   - For the official project announcement: [13, Chapman and Chang, 2000];
   - For `Bio.PDB`: [18, Hamelryck and Manderick, 2003];
   - For `Bio.Cluster`: [14, De Hoon *et al.*, 2004];
   - For `Bio.Graphics.GenomeDiagram`: [2, Pritchard *et al.*, 2006];
   - For `Bio.Phylo` and `Bio.Phylo.PAML`: [9, Talevich *et al.*, 2012];
   - For the FASTQ file format as supported in Biopython, BioPerl, BioRuby, BioJava, and EMBOSS: [7, Cock *et al.*, 2010].

2. *How should I capitalize "Biopython"? Is "BioPython" OK?*
   The correct capitalization is "Biopython", not "BioPython" (even though that would have matched BioPerl, BioJava and BioRuby).

3. *What is going wrong with my print commands?*
   This tutorial now uses the Python 3 style print *function*. As of Biopython 1.62, we support both Python 2 and Python 3. The most obvious language difference is the print *statement* in Python 2 became a print *function* in Python 3.

   For example, this will only work under Python 2:

   ```
   >>> print "Hello World!"
   Hello World!
   ```

   If you try that on Python 3 you'll get a `SyntaxError`. Under Python 3 you must write:

   ```
   >>> print("Hello World!")
   Hello World!
   ```

   Surprisingly that will also work on Python 2 – but only for simple examples printing one thing. In general you need to add this magic line to the start of your Python scripts to use the print function under Python 2.6 and 2.7:

   ```
   from __future__ import print_function
   ```

   If you forget to add this magic import, under Python 2 you'll see extra brackets produced by trying to use the print function when Python 2 is interpreting it as a print statement and a tuple.

4. *How do I find out what version of Biopython I have installed?*
   Use this:

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

If the "`import Bio`" line fails, Biopython is not installed. Note that those are double underscores before and after version. If the second line fails, your version is *very* out of date. If the version string ends with a plus, you don't have an official release, but a snapshot of the in development code.

5. *Where is the latest version of this document?*
   If you download a Biopython source code archive, it will include the relevant version in both HTML and PDF formats. The latest published version of this document (updated at each release) is online:

   - `http://biopython.org/DIST/docs/tutorial/Tutorial.html`
   - `http://biopython.org/DIST/docs/tutorial/Tutorial.pdf`

   If you are using the very latest unreleased code from our repository you can find copies of the in-progress tutorial here:

   - `http://biopython.org/DIST/docs/tutorial/Tutorial-dev.html`
   - `http://biopython.org/DIST/docs/tutorial/Tutorial-dev.pdf`

6. *What is wrong with my sequence comparisons?*
   There was a major change in Biopython 1.65 making the `Seq` and `MutableSeq` classes (and subclasses) use simple string-based comparison (ignoring the alphabet other than if giving a warning), which you can do explicitly with `str(seq1) == str(seq2)`.

   Older versions of Biopython would use instance-based comparison for `Seq` objects which you can do explicitly with `id(seq1) == id(seq2)`.

   If you still need to support old versions of Biopython, use these explicit forms to avoid problems. See Section 3.11.

7. *Why is the* `Seq` *object missing the upper & lower methods described in this Tutorial?*
   You need Biopython 1.53 or later. Alternatively, use `str(my_seq).upper()` to get an upper case string. If you need a Seq object, try `Seq(str(my_seq).upper())` but be careful about blindly re-using the same alphabet.

8. *Why doesn't the* `Seq` *object translation method support the* `cds` *option described in this Tutorial?*
   You need Biopython 1.51 or later.

9. *What file formats do* `Bio.SeqIO` *and* `Bio.AlignIO` *read and write?*
   Check the built in docstrings (`from Bio import SeqIO`, then `help(SeqIO)`), or see `http://biopython.org/wiki/SeqIO` and `http://biopython.org/wiki/AlignIO` on the wiki for the latest listing.

10. *Why won't the* `Bio.SeqIO` *and* `Bio.AlignIO` *functions* `parse`, `read` *and* `write` *take filenames? They insist on handles!*
    You need Biopython 1.54 or later, or just use handles explicitly (see Section 23.1). It is especially important to remember to close output handles explicitly after writing your data.

11. *Why won't the* `Bio.SeqIO.write()` *and* `Bio.AlignIO.write()` *functions accept a single record or alignment? They insist on a list or iterator!*
    You need Biopython 1.54 or later, or just wrap the item with `[...]` to create a list of one element.

12. *Why doesn't* `str(...)` *give me the full sequence of a* `Seq` *object?*
    You need Biopython 1.45 or later.

13. *Why doesn't* `Bio.Blast` *work with the latest plain text NCBI blast output?*
    The NCBI keep tweaking the plain text output from the BLAST tools, and keeping our parser up to date is/was an ongoing struggle. If you aren't using the latest version of Biopython, you could try upgrading. However, we (and the NCBI) recommend you use the XML output instead, which is designed to be read by a computer program.

14. *Why doesn't* `Bio.Entrez.parse()` *work? The module imports fine but there is no parse function!*
    You need Biopython 1.52 or later.

15. *Why has my script using* `Bio.Entrez.efetch()` *stopped working?*
    This could be due to NCBI changes in February 2012 introducing EFetch 2.0. First, they changed the default return modes - you probably want to add `retmode="text"` to your call. Second, they are now stricter about how to provide a list of IDs – Biopython 1.59 onwards turns a list into a comma separated string automatically.

16. *Why doesn't* `Bio.Blast.NCBIWWW.qblast()` *give the same results as the NCBI BLAST website?*
    You need to specify the same options – the NCBI often adjust the default settings on the website, and they do not match the QBLAST defaults anymore. Check things like the gap penalties and expectation threshold.

17. *Why doesn't* `Bio.Blast.NCBIXML.read()` *work? The module imports but there is no read function!*
    You need Biopython 1.50 or later. Or, use `next(Bio.Blast.NCBIXML.parse(...))` instead.

18. *Why doesn't my* `SeqRecord` *object have a* `letter_annotations` *attribute?*
    Per-letter-annotation support was added in Biopython 1.50.

19. *Why can't I slice my* `SeqRecord` *to get a sub-record?*
    You need Biopython 1.50 or later.

20. *Why can't I add* `SeqRecord` *objects together?*
    You need Biopython 1.53 or later.

21. *Why doesn't* `Bio.SeqIO.convert()` *or* `Bio.AlignIO.convert()` *work? The modules import fine but there is no convert function!*
    You need Biopython 1.52 or later. Alternatively, combine the `parse` and `write` functions as described in this tutorial (see Sections 5.5.2 and 6.2.1).

22. *Why doesn't* `Bio.SeqIO.index()` *work? The module imports fine but there is no index function!*
    You need Biopython 1.52 or later.

23. *Why doesn't* `Bio.SeqIO.index_db()` *work? The module imports fine but there is no* `index_db` *function!*
    You need Biopython 1.57 or later (and a Python with SQLite3 support).

24. *Where is the* `MultipleSeqAlignment` *object? The* `Bio.Align` *module imports fine but this class isn't there!*
    You need Biopython 1.54 or later. Alternatively, the older `Bio.Align.Generic.Alignment` class supports some of its functionality, but using this is now discouraged.

25. *Why can't I run command line tools directly from the application wrappers?*
    You need Biopython 1.55 or later. Alternatively, use the Python `subprocess` module directly.

26. *I looked in a directory for code, but I couldn't find the code that does something. Where's it hidden?*
    One thing to know is that we put code in `__init__.py` files. If you are not used to looking for code in this file this can be confusing. The reason we do this is to make the imports easier for users. For instance, instead of having to do a "repetitive" import like `from Bio.GenBank import GenBank`, you can just use `from Bio import GenBank`.

27. *Why does the code from CVS seem out of date?*
    In late September 2009, just after the release of Biopython 1.52, we switched from using CVS to git, a distributed version control system. The old CVS server will remain available as a static and read only backup, but if you want to grab the latest code, you'll need to use git instead. See our website for more details.

28. *Why doesn't* `Bio.Fasta` *work?*
    We deprecated the `Bio.Fasta` module in Biopython 1.51 (August 2009) and removed it in Biopython 1.55 (August 2010). There is a brief example showing how to convert old code to use `Bio.SeqIO` instead in the DEPRECATED file.

For more general questions, the Python FAQ pages `http://www.python.org/doc/faq/` may be useful.

Chapter 2

Quick Start – What can you do with Biopython?

This section is designed to get you started quickly with Biopython, and to give a general overview of what is available and how to use it. All of the examples in this section assume that you have some general working knowledge of Python, and that you have successfully installed Biopython on your system. If you think you need to brush up on your Python, the main Python web site provides quite a bit of free documentation to get started with (`http://www.python.org/doc/`).

Since much biological work on the computer involves connecting with databases on the internet, some of the examples will also require a working internet connection in order to run.

Now that that is all out of the way, let's get into what we can do with Biopython.

2.1 General overview of what Biopython provides

As mentioned in the introduction, Biopython is a set of libraries to provide the ability to deal with "things" of interest to biologists working on the computer. In general this means that you will need to have at least some programming experience (in Python, of course!) or at least an interest in learning to program. Biopython's job is to make your job easier as a programmer by supplying reusable libraries so that you can focus on answering your specific question of interest, instead of focusing on the internals of parsing a particular file format (of course, if you want to help by writing a parser that doesn't exist and contributing it to Biopython, please go ahead!). So Biopython's job is to make you happy!

One thing to note about Biopython is that it often provides multiple ways of "doing the same thing." Things have improved in recent releases, but this can still be frustrating as in Python there should ideally be one right way to do something. However, this can also be a real benefit because it gives you lots of flexibility and control over the libraries. The tutorial helps to show you the common or easy ways to do things so that you can just make things work. To learn more about the alternative possibilities, look in the Cookbook (Chapter 19, this has some cools tricks and tips), the Advanced section (Chapter 21), the built in "docstrings" (via the Python help command, or the API documentation) or ultimately the code itself.

2.2 Working with sequences

Disputably (of course!), the central object in bioinformatics is the sequence. Thus, we'll start with a quick introduction to the Biopython mechanisms for dealing with sequences, the `Seq` object, which we'll discuss in more detail in Chapter 3.

Most of the time when we think about sequences we have in my mind a string of letters like '`AGTACACTGGT`'. You can create such `Seq` object with this sequence as follows - the "`>>>`" represents the Python prompt followed by what you would type in:

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

What we have here is a sequence object with a *generic* alphabet - reflecting the fact we have *not* specified if this is a DNA or protein sequence (okay, a protein with a lot of Alanines, Glycines, Cysteines and Threonines!). We'll talk more about alphabets in Chapter 3.

In addition to having an alphabet, the `Seq` object differs from the Python string in the methods it supports. You can't do this with a plain string:

```
>>> my_seq
Seq('AGTACACTGGT', Alphabet())
>>> my_seq.complement()
Seq('TCATGTGACCA', Alphabet())
>>> my_seq.reverse_complement()
Seq('ACCAGTGTACT', Alphabet())
```

The next most important class is the `SeqRecord` or Sequence Record. This holds a sequence (as a `Seq` object) with additional annotation including an identifier, name and description. The `Bio.SeqIO` module for reading and writing sequence file formats works with `SeqRecord` objects, which will be introduced below and covered in more detail by Chapter 5.

This covers the basic features and uses of the Biopython sequence class. Now that you've got some idea of what it is like to interact with the Biopython libraries, it's time to delve into the fun, fun world of dealing with biological file formats!

2.3 A usage example

Before we jump right into parsers and everything else to do with Biopython, let's set up an example to motivate everything we do and make life more interesting. After all, if there wasn't any biology in this tutorial, why would you want you read it?

Since I love plants, I think we're just going to have to have a plant based example (sorry to all the fans of other organisms out there!). Having just completed a recent trip to our local greenhouse, we've suddenly developed an incredible obsession with Lady Slipper Orchids (if you wonder why, have a look at some Lady Slipper Orchids photos on Flickr, or try a Google Image Search).

Of course, orchids are not only beautiful to look at, they are also extremely interesting for people studying evolution and systematics. So let's suppose we're thinking about writing a funding proposal to do a molecular study of Lady Slipper evolution, and would like to see what kind of research has already been done and how we can add to that.

After a little bit of reading up we discover that the Lady Slipper Orchids are in the Orchidaceae family and the Cypripedioideae sub-family and are made up of 5 genera: *Cypripedium*, *Paphiopedilum*, *Phragmipedium*, *Selenipedium* and *Mexipedium*.

That gives us enough to get started delving for more information. So, let's look at how the Biopython tools can help us. We'll start with sequence parsing in Section 2.4, but the orchids will be back later on as well - for example we'll search PubMed for papers about orchids and extract sequence data from GenBank in Chapter 9, extract data from Swiss-Prot from certain orchid proteins in Chapter 10, and work with ClustalW multiple sequence alignments of orchid proteins in Section 6.4.1.