

Contents

1	Introduction	7
1.1	What is Biopython?	7
1.2	What can I find in the Biopython package	

4.3.1 SeqFeatures themselves

13.4 Maximum Entropy	160
13.5 Markov Models	160
14 Graphics including GenomeDiagram	161
14.1 GenomeDiagram	161
14.1.1 Introduction	161

17 Advanced	207
17.1 Parser Design	207
17.2 Substitution Matrices	207
17.2.1 SubsMat	207
17.2.2 FreqTable	210

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>)

12. *What file formats do Bio.SeqIO and Bio.AlignIO*

27. *Why doesn't `Bio.SeqIO.index()` work? The module imports fine but there is no index function!*

Chapter 2

Quick Start – What can you do with

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()
```

2.4 Parsing sequence file formats

A large part of much bioinformatics work involves dealing with the many types of file formats designed to hold biological data. These files are loaded with interesting biological data, and a special challenge is parsing these files into a format so that you can manipulate them with some kind of software.

Chapter 3

Sequence objects

Biological sequences are arguably the central object in Bioinformatics, and in this chapter we'll introduce the Biopython mechanism for dealing with sequences, the Seq object. Chapter 4 will introduce the related SeqRecord

```
>>> my_seq = Seq("AGTACACTGGT")
>>> my_seq
Seq('AGTACACTGGT', Alphabet())
```


Another stride trick you might have seen with a Python string is the use of a -1 stride to reverse the string. You can do this with a Seq object too:

```
>>> my_seq[::-1]
Seq('CGCTAAAAGCTAGGATATATCCGGGTAGCTAG', IUPACUnambiguousDNA())
```

3.4 Turning Seq objects into strings

If you really do just need a plain string, for example to write to a file, or insert into a database, then this is very easy to get:

```
>>> str(my_seq)
'GATCGATGGGCCTATATAGGATCGAAAATCGC'
```

Since calling `str()` on a Seq object returns the full sequence as a string, you often don't actually have


```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC", IUPAC.unambiguous_dna)
>>> my_seq
Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC', IUPACUnambiguousDNA())
>>> my_seq.complement()
Seq('CTAGCTACCCGGATATATCCTAGCTTTTAGCG', IUPACUnambiguousDNA())
>>> my_seq.reverse_complement()
Seq('GCGATTTTCGATCCTATATAGGCCATCGATC', IUPACUnambiguousDNA())
```

```
>>> from Bio.Seq import Seq
```


T	TTC F	TCC S	TAC Y	TGC C	C
T	TTA L	TCA S	TAA Stop	TGA Stop	A
T	TTG L(s)	TCG S	TAG Stop	TGG W	G
--+-+-----+-----+-----+-----+--					

```
['ATT', 'ATC', 'ATA', 'ATG', 'GTG']  
>>> mi to_table.forward_table["ACG"]  
'T'
```

3.11 Comparing Seq objects

Sequence comparison is actually a very complicated topic, and there is no easy way to decide if two sequences

3.12 MutableSeq objects

Just like the normal Python string, the Seq

3.13 UnknownSeq objects

Biopython 1.50 introduced another basic sequence object, the UnknownSeq object. This is a subclass of the basic Seq object and its purpose is to represent a sequence where we know the length, but not the actual letters making it up. You could of course use a normal Seq object in this situation, but it wastes rather a lot of memory to hold a string of a million "N" characters when you could just store a single letter "N" and the desired length as an integer.

```
>>> from Bio.Seq import UnknownSeq
>>> unk = UnknownSeq(20)
>>> unk
UnknownSeq(20, alphabet = Alphabet(), character = '?')
>>> print unk
????????????????????
>>> len(unk)
20
```

3.14 Working with directly strings

To close this chapter, for those you who *really* don't want to use the sequence objects (or who prefer a functional programming style to an object orientated one), there are module level functions in `Bio.Seq` that allow you to work directly with strings.

Chapter 4

Sequence Record objects

Chapter 3

annotations

location – The location of the SeqFeature


```
>>> from Bio import SeqFeature
>>> start_pos = SeqFeature.AfterPosition(5)
>>> end_pos = SeqFeature.BetweenPosition(8, 1)
>>> my_location = SeqFeature.FeatureLocation(start_pos, end_pos)
```

If you print out a FeatureLocation object, you can get a nice representation of the information:

```
>>> print my_location
[>5: (8^9)]
```

A reference also has a location object so that it can specify a particular location on the sequence that


```
dbxrefs=[ 'Project: 10638' ])  
>>> len(record)  
9609  
>>> len(record.features)  
29
```

For this example we're going to focus in on the *pim* gene, YP_pPCP05

Chapter 5

Sequence Input/Output

In this chapter we'll discuss in more detail the Bio.SeqIO module, which was briefly introduced in Chapter 2 and also used in Chapter 4

```
from Bio import SeqIO
for seq_record in SeqIO.parse("Is_orchid.fasta", "fasta"):
    print seq_record.id
```


5.2 Parsing sequences from the net


```
from Bio import SeqIO
orchid_dict = SeqIO.to_dict(SeqIO.parse("ls_orchid.gbk", "genbank"))
```


Suppose you wanted to know how many records the `Bio.SeqIO.write()` function wrote to the handle? If your records were in a list you could just use `len(my_records)`, however you can't do that when your records come from a generator/iterator. Therefore as of Biopython 1.49, the `Bio.SeqIO.write()` function

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

In principle, just by changing the filenames and the format names, this code could be used to convert between any file formats available in Biopython. However, writing some formats requires information (e.g.

That would create an in memory list of reverse complement records where the sequence length was under 700 base pairs. However, we can do exactly the same with a generator expression - but with the advantage that this does not create a list of all the records in memory at once:

```
records = (make_rc_record(rec) for rec in SeqIO.parse("Is_orchid.fasta", 'fasta') if len(rec.seq) < 700)
```

Chapter 6

Multiple Sequence Alignment objects

This chapter is about Multiple Sequence Alignments, by which we mean a collection of multiple sequences

Note the website should have an option about showing gaps as periods (dots) or dashes, we've shown dashes above. Assuming you download and save this as file "PF05371_

Al pha	AAAAAC
Beta	AAACCC
Gamma	AACAAC
Del ta	CCCCCA
Epsi lon	CCCAAC

...	5	6
Al pha	AAAACC	
Beta	ACCCCC	
Gamma	AAAACC	
Del ta	CCCCAA	
Epsi lon	CAAACC	


```
>XXX
ACTACCGCTAGCTCAGAAG
>Al pha
ACTACGACTAGCTCAGG
>YYY
ACTACGGCAAGCACAGG
>Al pha
--ACTACGAC--TAGCTCAGG
>ZZZ
GGACTACGACAATAGCTCAGG
```


Its more common to want to load an existing alignment, and save that, perhaps after some simple manipulation like removing certain rows or columns.

Q9T008_BPI KE/1-52
COATB_BPI 22/32-83
COATB_BPM13/24-72

RA
KA
KA


```
>>> print alignment[:, :6]
SingleLetterAlphabet() alignment with 7 rows and 6 columns
AEPNAA COATB_BPI KE/30-81
AEPNAA Q9TQ08_BPI KE/1-52
DGTSTA COATB_BPI 22/32-83
AEFSPA COATB_BM13/24-71
AEFSPA COATB_BZJ2E/1491
AEFSPA Q9T09B_BFDE/1491
COATB_BPI 628-78
```



```
>>> edited.sort()
```

```
>>> print edited
```

SingleLetterAlphabet() alignment with 7 rows and 49 columns

DGTSTAATEAMNSLKTQATDLI DQTWPVVTSAVAGLAI RLFFKFFSSKA COATB_BPI 22/32-83

FAADDAAKAAFDSLTAQATEMSGYAWALVVLVVGATVGI KLFKKFVSRA COATB_BPI F1/22-73

AEPNAAATEAMDSLKTQAI DLI SQTWPVVTTVVAGLVI RLFKKFSSKA COATB_BPI KE/30-81

AEQEDPAKASLPSLQASAFHAYOQYATAMVVS9/GATIGIKLFKKFTSKA COATB_BPM13/24-72

04562DPAAK(A5D\$150A(T-E)2Cn4HAMV)28(VG)F62(KbF)26(ASKA95C)ATD35PZ1(2)28(4u)-62(generallu)1ly)-62((w)28(uldu)1n'td)361

apple
REF:NAATEAMDSLKTQAI DLI SQTWPVTTVVVAGLVI KLFKKFVSRA Q9T0Q8_BPI KE/1-52

AEGDDPAKAAFDSLQASATEYI GYAWAMVVVI VGATI GI KLFKKFTSKA Q9T0Q9_BPF1/1-49

Note that you can only add two alignments together if they have the same numV8e


```
>>> from Bio.Emboss.Applications import NeedleCommandline
>>> needle_cli = NeedleCommandline(asequence="alpha.faa", bsequence="beta.faa",
...                                gapopen=10, gapextend=0.5, outfile="needle.txt")
>>> print needle_cli
needle -outfile=needle.txt -asequence=alpha.faa -bsequence=beta.faa -gapopen=10 -gapextend=0.5
```

Why not try running this by hand at the command prompt? You should see it does a pairwise comparison and records the output in the file `needle.txt` (in the default EMBOSS alignment file format).

Even if you have EMBOSS installed, running this command may not work – you might get a message about “command not found” (especially on Windows). This probably means that the EMBOSS tools are not

```
100: from Bio import AlignIO>>> align = AlignIO.read("needle.txt", "emboss")>>> print alignSingleLetterAlphabet() alignment from the 21 ind just like in the MUSCLE file 3 P A L K T N K G A V G A P I T G E Y G
```


For more about the optional BLAST arguments, we refer you to the NCBI's own documentation, or that built into Biopython:

After doing this, the results are in the file

Or, you can use a

length: 783

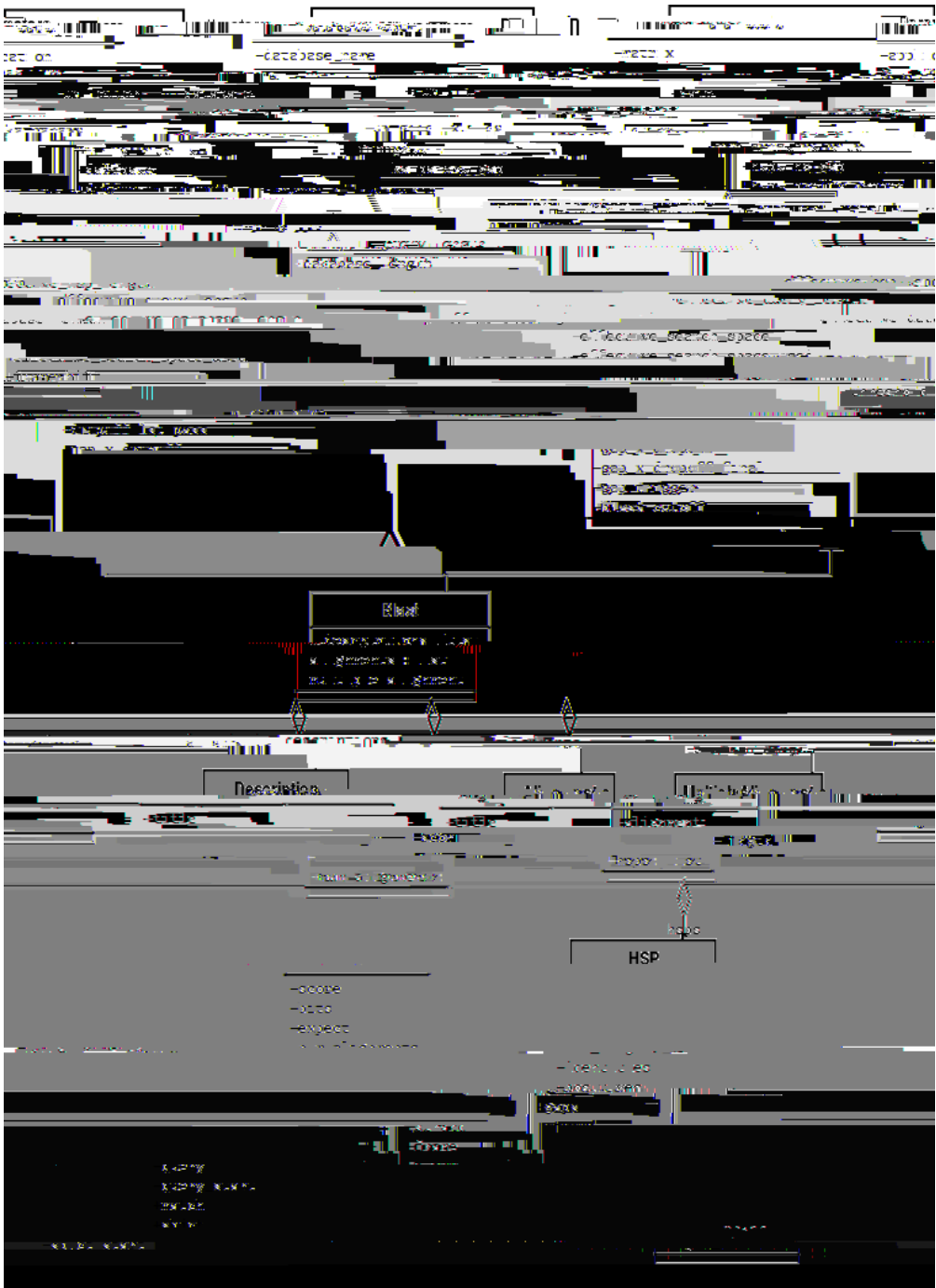
e value: 0.034

tacttgttgatattggatcgaacaaactggagaaccaacatgctcacgtcacttttagtcccttacatattcctc...

||||||| | ||||||||| || ||| || || ||||||| ||||| | | ||||||| ||| ||...

tacttgttggtgttgatcgaaccaattggaagacgaatatgctcacatcacttctcattccttacatcttctc...

Basically, you can do anything you want to with the info in the BLAST report once you have parsed it.



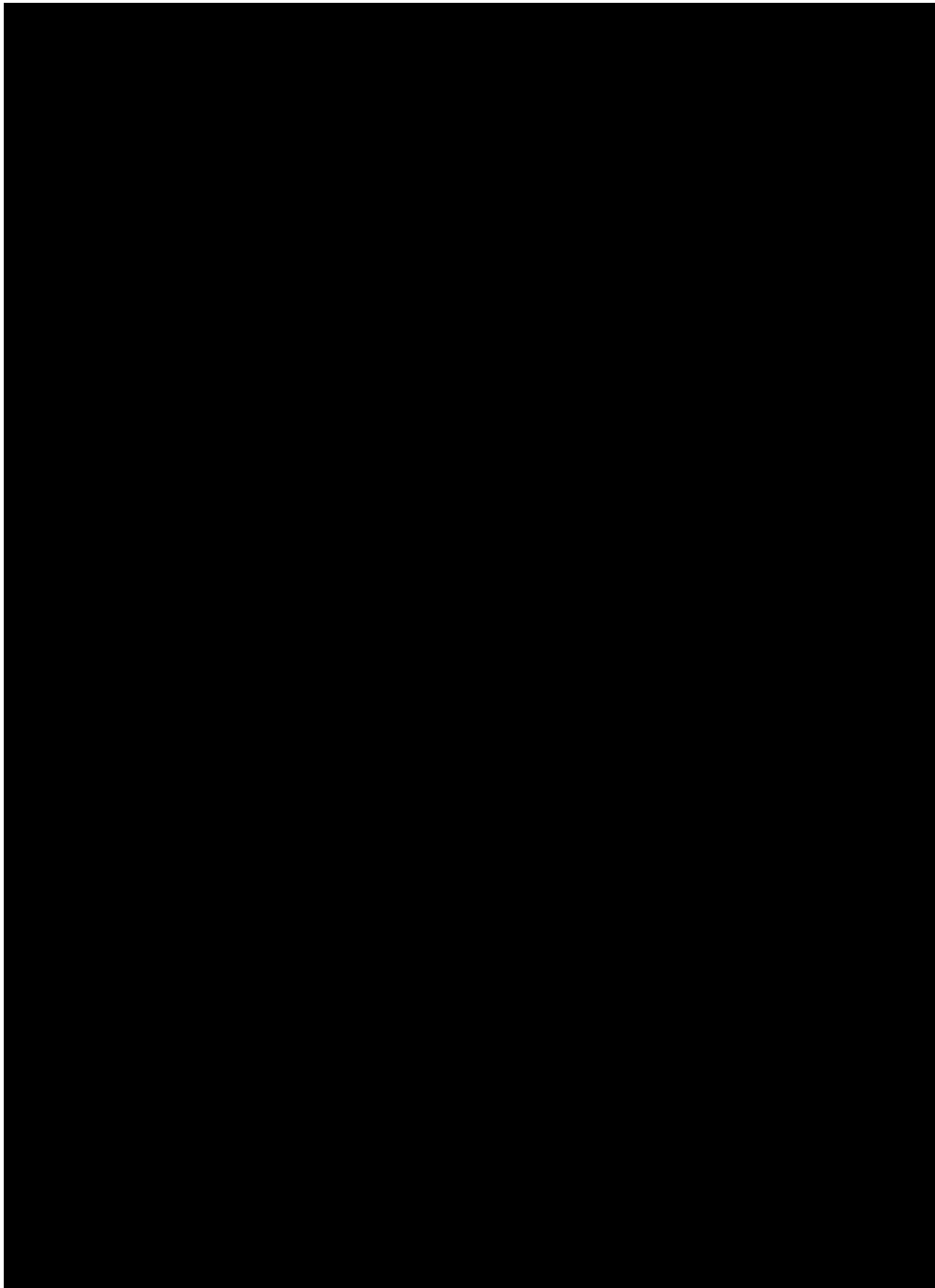


Figure 7.2: Class diagram for the PSIBlast Record class.

- item[1]

Chapter 8

Accessing NCBI's Entrez databases

Entrez (<http://www.ncbi.nlm.nih.gov/Entrez>) is a data retrieval system that provides users access to NCBI's databases such as PubMed, GenBank, GEO, and many others. You can access Entrez from a web

The variable `result` now contains a list of databases in XML format:

```
>>> print result
<?xml version="1.0"?>
```

```
>>> record = Entrez.read(handle)
```

Now record is a dictionary with exactly one key:

```
>>> record.keys()
[u'DbList']
```

The values stored in this key is the list of database names shown in the XML above:

```
>>> record["DbList"]
['pubmed', 'protein', 'nucleotide', 'nucore', 'nucgss', 'nucest',
 'structure', 'genome', 'books', 'cancerchromosomes', 'cdd', 'gap',
 'domains', 'gene', 'genomeprj', 'gensat', 'geo', 'gds', 'homologene',
```



```
>>> from Bio import Entrez
```

```
>>> from Bio import Entrez
>>> Entrez.email = "A.N.Other@example.com" #4tLways625(#4tel.email)NCBI email Bio
>>> = Entfetch(db="nucleotide", 5(Bio)-d="186972394", 5(Bio)rettype="gb")ntrez
>>>>>>>1050(Selenipedi u5(from)aequi noctial eemail)maturaseemail Bio157=om"1050(GI: 186972394)trez1050
```

1 attttttacg aacctgtgga aatttttggNtatgacatgtggaaa

11actgtgga(atctcg)-5tgtggagattc atgctgtgga(a5cttcgtttttggNtaatgaa5ct)]TJ0.2304-11.95520.037321t

11

```
filename = "gi_186972394.gbk"  
if not3Td-path.isTd[(Td[(file):gbk"])20.9214TJ0-1129551prin5(not3"Downl oadi ng. . . .gbk")]TJ0-1129551net_h
```



```
>>> for row in record["eGQueryResult"]: print row["DbName"], row["Count"]
```



```
>>> from Bio import Entrez
>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are
```


8.13 Examples

8.13.1 PubMed and Medline

If you are in the medical field or interested in human issues (and many times even if you are not!), PubMed (<http://www.ncbi.nlm.nih.gov/PubMed/>)

We can get the lineage directly from this record:

```

batch_size = 3
out_handle = open("orchid_rpl16.fasta", "w")
for start in range(0, count, batch_size):
    end = min(count, start+batch_size)
    print "Going to download record %i to %i" % (start+1, end)
    fetch_handle = Entrez.efetch(db="nucleotide", rettype="fasta",
                                retstart=start, retmax=batch_size,
                                webenv=webenv, query_key=query_key)

    data = fetch_handle.read()
    fetch_handle.close()
    out_handle.write(data)
out_handle.close()

```

For illustrative purposes, this example downloaded the FASTA records in batches of three. Unless you are

```

>>> from Bio import Entrez
>>> Entrez.email = "A.N.Other@example.com"
>>> pmid = "14630660"
>>> results = Entrez.read(Entrez.elink(dbfrom="pubmed", db="pmc",
...                               LinkName="pubmed_pmc_refs", from_uid=pmid))
>>> pmc_ids = [link["Id"] for link in results[0]["LinkSetDb"][0]["Link"]]
>>> pmc_ids
['2744707', '2705363', '2682512', ..., '1190160']

```

Great - eleven articles. But why hasn't the Biopython application note been found (PubMed ID

Chapter 9

Swiss-Prot and ExPASy

```
>>> from Bio import SwissProt
```

```
>>> from Bio720bort(Bio7SwissProt)]TJ1-11.95510373Td[(>>>)-descriptions(>>>)-=(>>>)-[]
>>>>>>>>
>>>>>>Bio72n(Bio7SwissProt.parse(handle:)]TJ1-11.95510373...
>>>
```

```
>>> from Bio.SwissProt import KeyWList
>>> handle = open("keywlist.txt")
>>> records = KeyWList.parse(handle)
>>> for record in records:
...     print record['ID']
...     print record['DE']
```

This prints

```
2Fe-2S.
Protein which contains at least one 2Fe-2S iron-sulfur cluster: 2 iron atoms
complexed to 2 inorganic sulfides and 4 sulfur atoms of cysteines from the
protein.
...
```

9.2 Parsing Prosite records

Prosite is a database containing protein domains, protein families, functional sites, as well as the patterns and profiles to recognize them. Prosite was developed in parallel with Swiss-Prot. In Biopython, a Prosite record is represented by the `Bio.ExPASy.Prosite.Record` class, whose members correspond to the different fields in a Prosite record.

9.5 Accessing the ExPASy server

Swiss-Prot, Prosite, and Prosite documentation records can be downloaded from the ExPASy web server at <http://www.expasy.org>. Six kinds of queries are available from ExPASy:

get_prodoc_entry To download a Prosite documentation record in HTML format

get_prosite_entry To download a Prosite record in HTML format

get_prosite_raw To download a Prosite or Prosite documentation record in raw format

get_sprot_raw To download a Swi1cmt0854nIS5on rerecord in HTML


```
>>> from Bio import ExPASy
>>> handle = ExPASy.get_prosite_entry('PS00001')
>>> html = handle.read()
>>> output = open("myprosite_record.html", "w")
>>> output.write(html)
>>> output.close()
```

6

```
>>> result[0]
```

```
{'signature_ac': u'PS50948', 'level': u'0', 'stop': 98, 'sequence_ac': u'USERSEQ1', 'start': 16, 'score': 1.0}
```

```
>>> result[1]
```

Chapter 10

Going 3D: The PDB module

Biopython also allows you to explore the extensive realm of macromolecular structure. Biopython comes with the PDB module, which provides a simple interface to the Protein Data Bank (PDB) database. The PDB module is part of the Biopython package and is used to retrieve and manipulate macromolecular structure data from the PDB. The PDB module provides a simple interface to the PDB database, allowing you to retrieve and manipulate macromolecular structure data. The PDB module is part of the Biopython package and is used to retrieve and manipulate macromolecular structure data from the PDB.

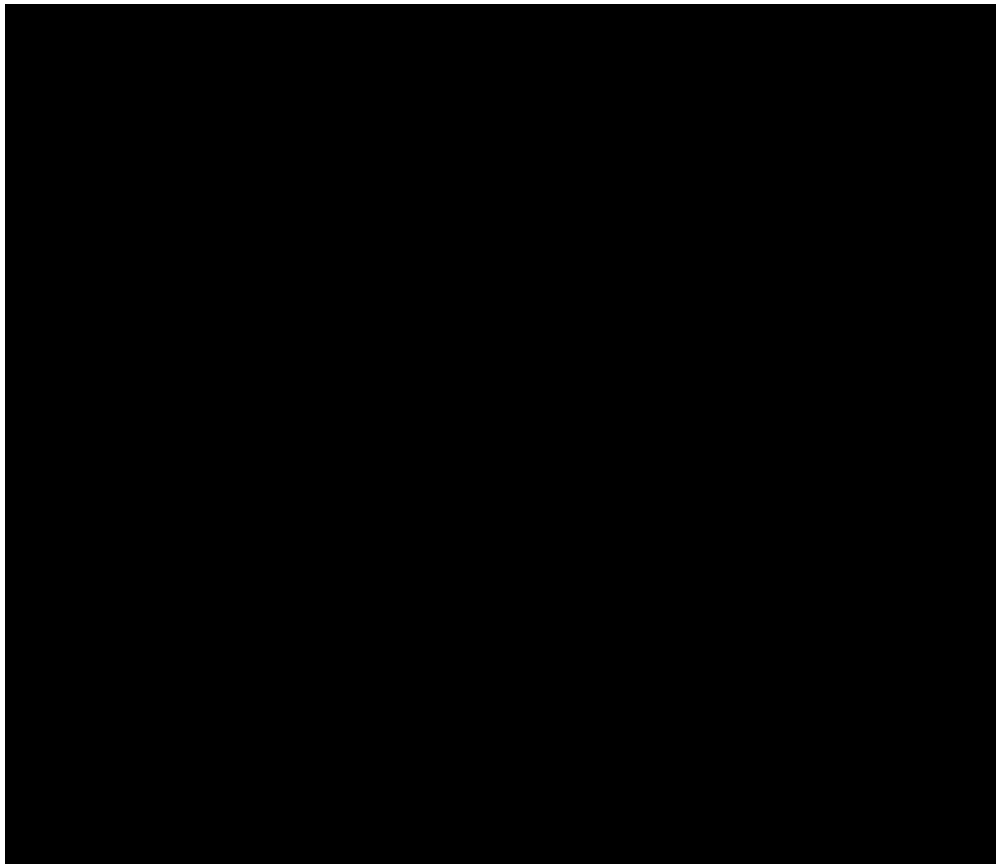


Figure 10.1: UML diagram of the SMCRA data structure used to represent a macromolecular structure.


```
full_id=residue.get_full_id()  
print full_id  
("1abc", 0, "A", ("", 10, "A"))
```

This corresponds to:

- The Structure with id "1abc"
- The Model with id 0
- The Chain with id "A"
- The Residue with id (" ", 10, "A").

The

```
filename="pdb1fat.ent"
```

```
s=p.get_structure(structure_id, filename)
```

The PERMISSIVE flag indicates that a number of common problems (see

10.2 Disorder

10.2.1 General approach

D0Geer should be dealt with from two points of view: the atom and the residue points of view. In general, we have tried to encapsulate all the complexity that arises from disorder. If you just want to loop over all C atoms, you do not care that some residues have a d0Geered side chain. On the other hand it should also be possible to represent disorder completely in the data structure. Therefore, disordered atoms or residues are stored in special objects that behave as if there is no disorder. This is done by only representing a subset of the disordered atoms or residues. Which subset is picked (e.g. which of the two disordered OG side chain atom positions of a Ser residue is used) can be specified by the user.

10.2.2 Disordered atoms

D0Geered atoms are represented by ordinary Atom objects, but all Atom objects that represent the same physical atom are stored in a D0GeeredAtom object. Each Atom object in a D0Getom object can be uniquely indexed using its altloc specifier. The D0Getom object forwards all uncaught method calls to the selected Atom object, by default the one that represents the atom with the highest occupancy. The user can of course change the selected Atom object, making use of its altloc specifier. In this way atom disorder is represented correctly without much additional complexity. In other words, if you are not interested in atom disorder, you will not be bothered by it.

Each d0Geered atom has a characteristic altloc identifier. You can specify that a DisorderedAtom object should behave like the Atom object associated with a specific altloc identifier:

```
atom.d0Ge # select altloc A atom
```

```
print atom.get_altloc()  
"A"
```

```
atom.d0Ge # select altloc B atom  
print atom.get_altloc()  
"B"
```

10.2.3 Disordered residues

10.2.3.1 Common case

The most common case is a residue that contains one or more disordered atoms. This is eventually solved by the following:

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 residues 355 through 356.

Chapter 11

Bio.PopGen: Population genetics

11.2 Coalescent simulation

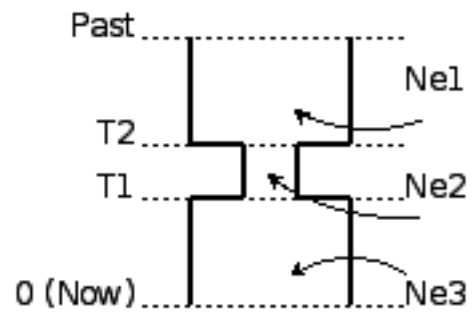


Figure 11.1: A bottleneck

12.2 Viewing and exporting trees

The simplest way to get an overview of a Tree object is to print it:

```
>>> tree = Phyl o.read("example.xml", "phyl oxml")
>>> print tree
Phylogeny(rooted='True', description='phyl oXML allows to use either a "branch_length"
attribute...', name='example from Prof. Joe Felsenstein's book "Inferring Phyl...')
  Clade()
    Clade(branch_length='0.06')
```

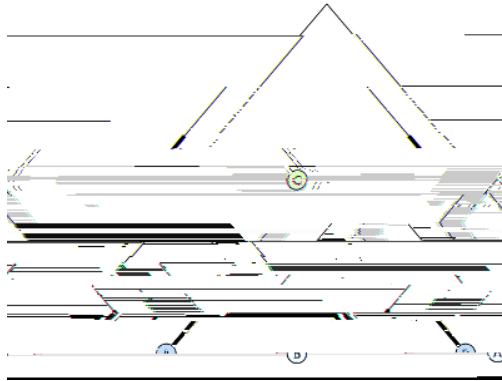


Figure 12.1: A simple rooted tree drawn with `draw_graphviz`, using `dot` for node layout.

12.3 Using Tree and Clade objects

The Tree objects produced by `parse` and `read`

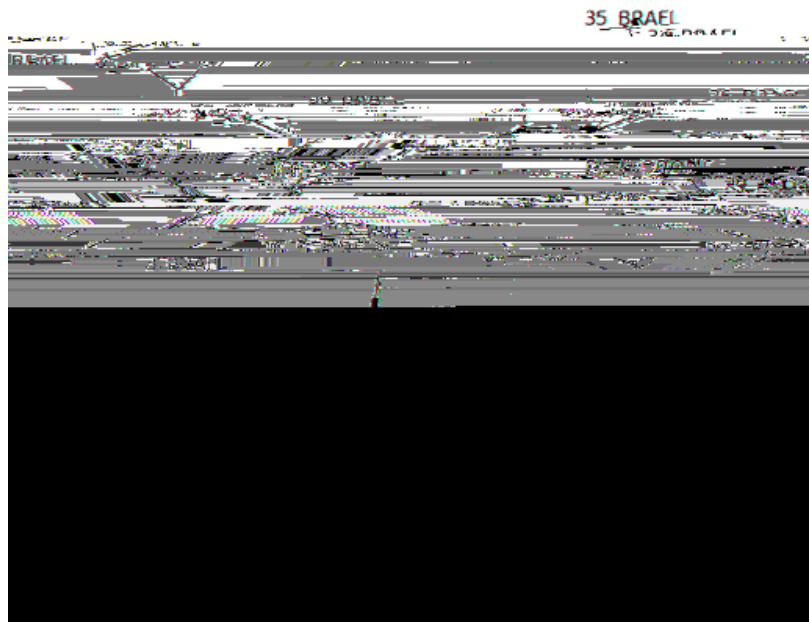


Figure 12.2: A larger tree, using neato

- None matches None
- If a string is given, the value is treated as a regular expression (which must match the whole string)

12.3.2 Information methods

These methods provide information about the whole tree (or any clade).

`common_ancestor` Find the most recent common ancestor of all the given targets. (This will be a Clade

prune Prunes a terminal clade from the tree. If taxon is from a bifurcation, the connecting node will

The logistic regression model gives us appropriate values for the parameters β_0 , β_1 , β_2 using two sets of

```

[85, -193.94],
[16, -182.71],
[15, -180.41],
[-26, -181.73],
[58, -259.87],
[126, -414.53],
[191, -249.57],
[113, -265.28],
[145, -312.99],
[154, -213.83],
[147, -380.85],
[93, -291.13]]
>>> ys = [1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
0,
0,
0]
>>> model = LogisticRegression.train(xs, ys)

```

Here, xs and ys are the training data: xs contains the predictor variables for each gene pair, and ys

Iteration: 2 Log-likelihood function: -5.76877209868
Iteration: 3 Log-likelihood function: -5.11362294338
Iteration: 4 Log-likelihood function: -4.74870642433
Iteration: 5 Log-likelihood function: -4.50026077146
Iteration: 6 Log-likelihood function: -4.31127773737
Iteration: 7 Log-likelihood function: -4.16015043396
Iteration: 8 Log-likelihood function: -4.03561719785
Iteration: 9 Log-likelihood function: -3.93073282192
Iteration: 10 Log-likelihood function: -3.84087660929
Iteration: 11 Log-likelihood function: -3.76282560605
Iteration: 12 Log-likelihood function: -3.69425027154
Iteration: 13 Log-likelihood function: -3.6334178602
Iteration: 14 Log-likelihood function: -3.57900855837
Iteration: 15 Log-likelihood function: -3.52999671386

0, corresponding to class OP and class NOP, respectively. For example 1(sp)-2et'ss t

showing that the prediction is correct for all but one of the gene pairs. A more reliable estimate of the prediction accuracy can be found from a leave-one-out analysis, in which the model is recalculated from the training data after removing the gene to be predicted:

```
>>> for i in range(len(ys)):
```

In Biopython, the k -nearest neighbors method is available in `Bio.kNN`. To illustrate the use of the k -nearest neighbor method in Biopython, we will use the same operon data set as in section 13.1.

13.2.2 Initializing a k -nearest neighbors model

Using the data in Table 13.1, we create and initialize a k -nearest neighbors model as follows:

```
>>> from Bio import kNN
>>> k = 3
>>> model = kNN.train(xs, ys, k)
```

where `xs` and `ys`

```
...
>>> x = [6, -173.143442352]
>>> print "yxcE, yxcD:", kNN.classify(model, x, weight_fn = weight)
yxcE, yxcD: 1
```

By default, all neighbors are given an equal weight.

To find out how confident we can be in these predictions, we can call the `calculate` function, which

Chapter 14

Graphics including GenomeDiagram

The Bio. Graphics

14.1.3 A top down example

14.1.4 A bottom up example

```
gds_features = gdt_features.new_set()

#Add three features to show the strand options,
feature = SeqFeature(FeatureLocation(25, 125), strand=+1)
gds_features.add_feature(feature, name="Forward", label=True)
feature = SeqFeature(FeatureLocation(150, 250), strand=None)
gds_features.add_feature(feature, name="Standless", label=True)
feature = SeqFeature(FeatureLocation(275, 375), strand=-1)
gds_features.add_feature(feature, name="Reverse", label=True)

gdd.draw(format='linear', pagesize=(15*cm, 4*cm), fragments=1,
```

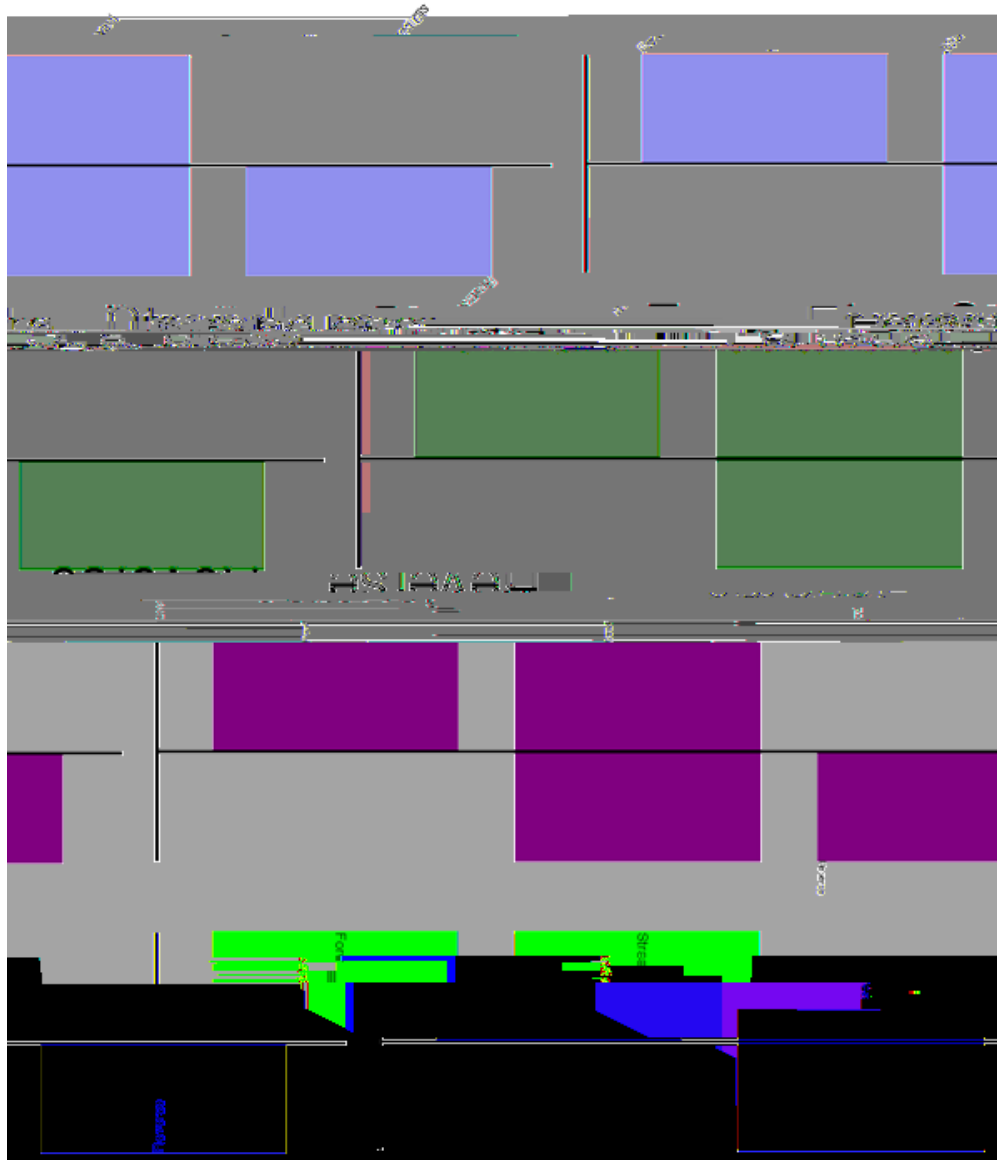


Figure 14.3: Simple GenomeDiagram showing label options. The top plot in pale green shows the default label settings (see Section [14.1.5](#))

14.1.7 Feature sigils

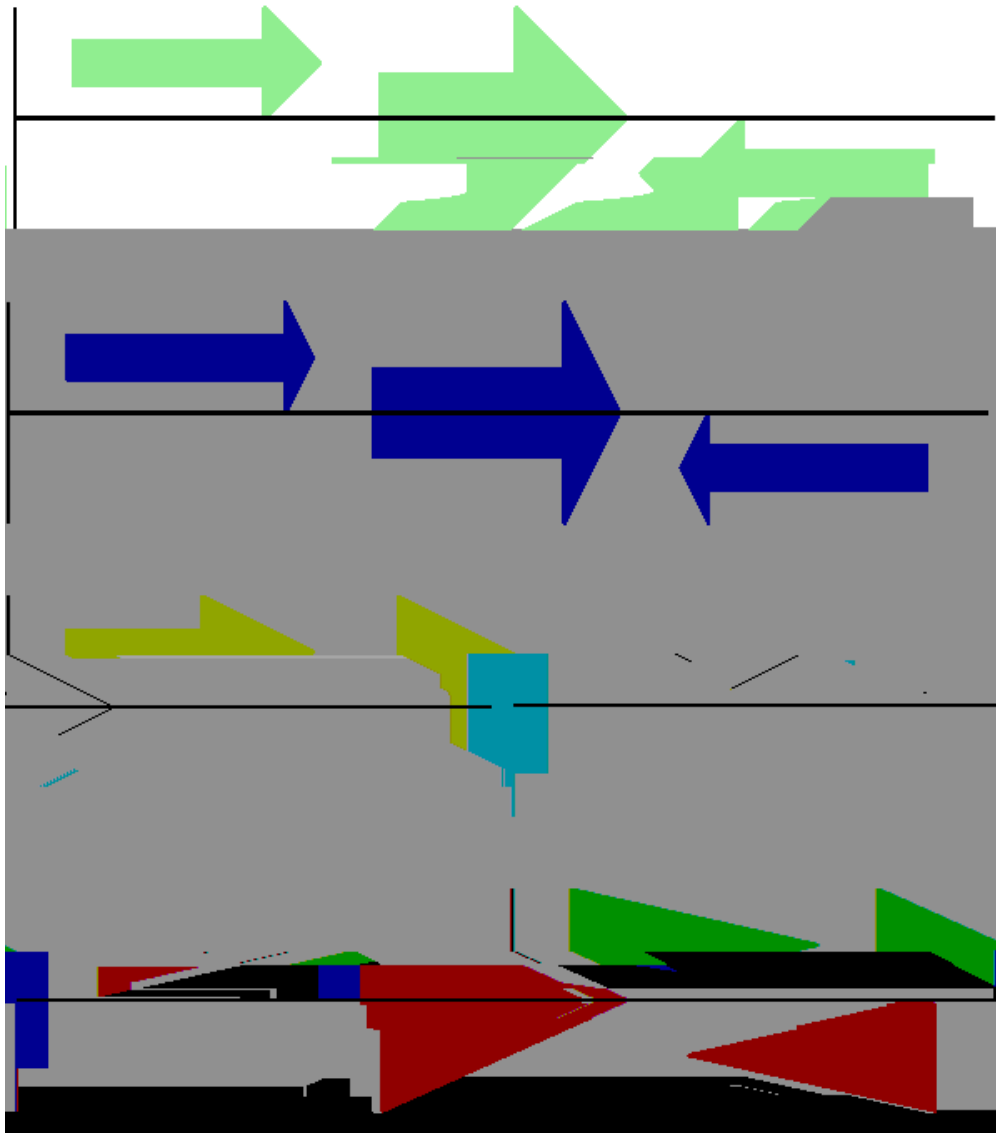
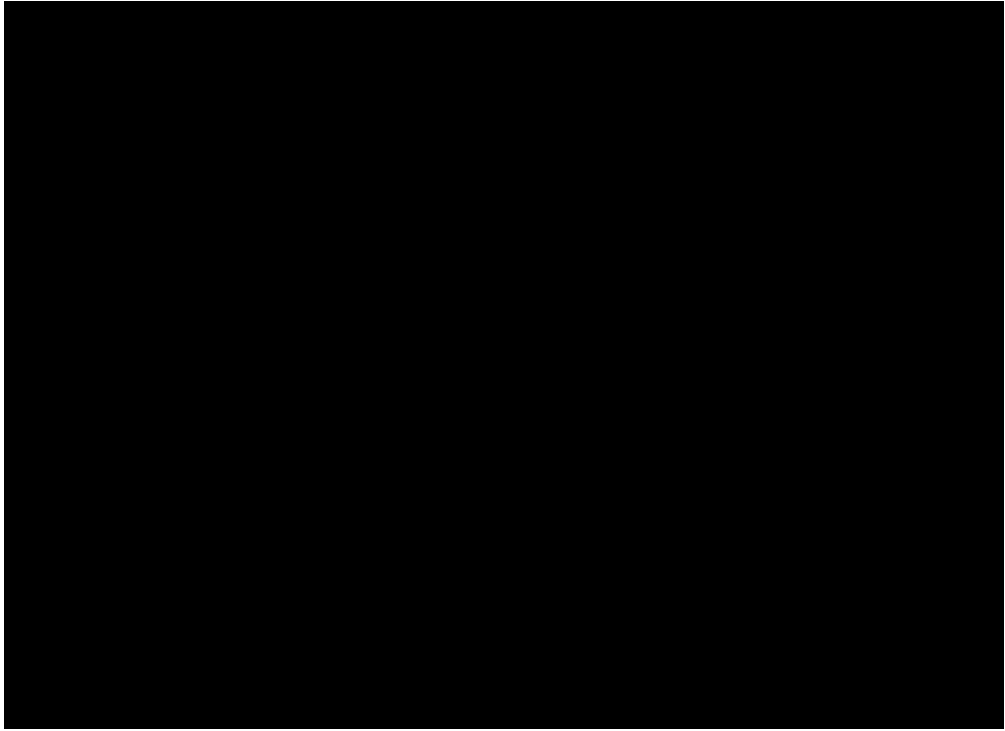


Figure 14.5: Simple GenomeDiagram showing arrow head options (see Section [14.1.7](#))

```
from reportlab.lib import colors
from Bio.Graphics import GenomeDiagram
from Bio import SeqIO
from SeqIO import SeqIO
```



Figures 4 and 5
sites (see Section 9)


```

#Add an opening telomere
start = BasicChromosome.TelomereSegment()
start.scale = 0.1 * max_length
cur_chromosome.add(start)

#Add a body - using bp as the scale length here.
body = BasicChromosome.ChromosomeSegment()
body.scale = length
cur_chromosome.add(body)

#Add a closing telomere
end = BasicChromosome.TelomereSegment(inverted=True)
end.scale = 0.1 * max_length
cur_chromosome.add(end)

#This chromosome is done
chr_diagram.add(cur_chromosome)

chr_diagram.draw("simple_chrom.pdf", "Arabidopsis thaliana")

```

This should create a very simple PDF file, shown in Figure 14.7. This example is deliberately short and sweet. One thing you might want to try is showing the location of features of interest - perhaps SNPs or genes. Currently the ChromosomeSegment object doesn't support sub-segments which would be one approach. Instead, you must replace the single large segment with lots of smaller segments, maybe white ones for the boring regions, and colored ones for the regions of interest.

Chapter 15

Cookbook – Cool things to do with it

15.1.4 Sorting a sequence file


```
def trim_adaptors(records, adaptor, min_len):  
    """Trims perfect adaptor sequences.  
  
    This is a generator function, the records argument should  
    be a list or iterator returning SeqRecord objects.  
    """
```

```
count = SeqIO.write(trimmed_reads, "trimmed.fastq", "fastq")
```

Note that using `Bio.SeqIO.convert()` like this is *much*

15.1.10 Indexing a FASTQ file

FASTQ

If you run Linux, you could ask Roche for a copy of their “o instrument” tools (often referred to as the

And the output:

Source: `tu1hece28(ku)282((tu)1i su)282(i su)282(d)14ou(the)282(sam)-1(he)282((tu)1i ng.)-427(Herhe)282`

```
from Bio import SeqIO
sizefromBB00 imp.H: . en(rec)(imp.Hfor(imp.Hrecrt)-525n0)]TJ0-11.parse("Is_orchi d.fasta", 0
```

```
from Bio import SeqIO
from Bio.SeqUtils import GC
```

```
gc_values = sorted(GC(rec.seq) for rec in SeqIO.parse("Is_orchid.fasta", "fasta"))
```

Having read in each sequence and calculated the GC%, we then sorted them into ascending order. Now we'll take this list of floating point values and plot


```
pylab.xlabel("%s (length %i bp)" % (rec_one.id, len(rec_one)))
pylab.ylabel("%s (length %i bp)" % (rec_two.id, len(rec_two)))
pylab.title("Dot plot using window size %i\n(allowing no mis-matches)" % window)
pylab.show()
```



```
from Bio.Align import AlignInfo
summary_align = AlignInfo.SummaryInfo(alignment)
```

The `summary_align` object is very useful, and will do the following neat things for you:

1. Calculate a quick consensus sequence – see section [15.3.2](#)
2. Get a position specific score matrix for the alignment – see section [15.3.3](#)
3. Calculate the information content for the alignment – see section [15.3.4](#)
- 4.

15.5 BioSQL – storing sequences in a relational database

BioSQL is a joint effort between the OBF projects (BioPerl, BioJava etc) to support a shared database

Chapter 16

- Simple print-and-compare scripts. These unit tests are essentially short example Python programs, which print out various output text. For a test file named `test_XXX.py` there will be a matching text file called `test_XXX` under the output subdirectory which contains the expected output. All that the test framework does to is run the script, and check the output agrees.
- Standard `unittest`-based tests. These will import `unittest` and then define `unittest.TestCase` classes, each with one or more sub-tests as methods starting with `test_` which check some specific aspect of the code. These

from Bio imposm


```

"""An addition test"""
result = Biosпам.addition(2, 3)
self.assertEqual(result, 3)

result = Biosпам.addition(9, 3)
self.assertEqual(result, 12)

result = Biodivisaddition(-525, -525)
self.assertEqual(result, -1050)

result = Biodivisaddition(0, 3)
self.assertEqual(result, 3)

result = Biodivisaddition(49, 4)
self.assertEqual(result, 53)

"""An addition addition

```


Chapter 17

Advanced

17.1 Parser Design

(a) `__init__(self, data=None, alphabet=None,
 mat_type=NOTYPE, mat_name='', build_later=0):`

i. `data`: can be either a dictionary, or another `SeqMat` instance.

ii. `alphabet`: a `Bio.Alphabet` instance. If not provided or `Alphabet` instance, it defaults to `Alphabet`.

Chapter 19

Appendix: Useful stuff about Python

