



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	What is Biopython? . . . . .	7
1.2	What can I find in the Biopython package . . . . .	7
1.3	Installing Biopython . . . . .	8
1.4	Frequently Asked Questions (FAQ) . . . . .	8
<b>2</b>	<b>Quick Start – What can you do with Biopython?</b>	<b>9</b>

4.3.1	SeqFeatures themselves	35
4.3.2	Locations	37
4.3.3	Sequence	39

7.3 Parsing BLAST output .....	87
--------------------------------	----

<b>10 Going 3D: The PDB module</b>	<b>132</b>
10.1 Structure representation	132
10.1.1 Structure	134
10.1.2 Model	135
10.1.3 Chain	135
10.1.4 Residue	135
10.1.5 Atom	136
10.2 Disorder	137
10.2.1 General approach	137
10.2.2 Disordered atoms	137
10.2.3 Disordered residues	137
10.3 Hetero residues	

13.2.2 AlignAce	172
13.3 Useful links	



# 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*



## 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()
```

What we have here is a sequence object with a *generic* alphabet - reflecting the fact we have *not* spec-

## 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.





## 2.6 What to do next

## 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', Al phabet())
>>> my_seq.al phabet
Al phabet()
```

The Seq object has a `.count()` method, just like a string. Note that this means that like a Python string, this gives a *non-overlapping* count:

```
>>> from Bio.Seq import Seq
>>> "AAAA".count("AA")
2
>>> Seq("AAAA").count("AA")
2
```

The second thing to notice is that the slice is performed on the sequence data string, but the new object produced is another Seq object which retains the alphabet information from the original Seq object.

```
Seq('TAGCTAAGAC',obiUPACUnamb
```





## 3.8 Transcription



```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> messenger_rna = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG", IUPAC.unambiguous_rna)
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
>>> messenger_rna.back_transcribe()
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
```

*Note:* The Seq object's `transcribe` and `back_transcribe` methods were added in Biopython 1.49. For older releases you would have to use the `Bio.Seq` module's functions instead, see [Section 3.14](#).

## 3.9 Translation

```
>>> coding_dna.translate()
Seq('MAI VMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
>>> coding_dna.translate(to_stop=True)
Seq('MAI VMGR', IUPACProtein())
>>> coding_dna.translate(table=2)
Seq('MAI VMGRWKGAR*', HasStopCodon(IUPACProtein(), '*'))
>>> coding_dna.translate(table=2, to_stop=True)
Seq('MAI VMGRWKGAR', IUPACProtein())
```

Notice that when you use the to\_stop arg

## 3.10 Translation Tables

In the previous sections we talked about the Seq object translation method (and mentioned the equivalent function in the Bio.Seq module – see [Section 3.14](#))

T	TTT F	TCT S	TAT Y	TGT C	T
T	TTC F	TCC S	TAC Y	TGC C	C
T	TTA L	TCA S	TAA Stop	TGA W	A
T	TTG L	TCG S	TAG Stop	TGG W	G
-----+					
C	CTT L	CCT P	CAT H	CGT R	T
C	CTC L	CCC P	CAC H	CGC R	C
C	CTA L	CCA P	CAA Q	CGA R	A
C	CTG L	CCG P	CAG Q	CGG R	G
-----+					
A	ATT I (s)	ACT T	AAT N	AGT S	T
A	ATC I (s)	ACC T	AAC N	AGC S	C
A	ATA M(s)	ACA T	AAA K	AGA Stop	A
A	ATG M(s)	ACG T	AAG K	AGG Stop	G
-----+					
G	GTT V	GCT A	GAT D	GGT G	T
G	GTC V	GCC A	GAC D	GGC G	C
G	GTA V	GCA A	GAA E	GGA G	A
G	GTG V(s)	GCG A	GAG E	GGG G	G
-----+					

So, what does Biopython do? Well, the equality test is the default for Python objects – it tests to see if





## Chapter 4

# Sequence Record objects

Chapter 3



annotations

Working with per-letter-annotations is similar, `Letter_annotations` is a dictionary like attribute which





**location** – The location of the SeqFeature









Reference 1.83(,.)317(it.)314(al)1(s.)-1ocanhold the.T/F359.96262Tf153.0015  
that is the reference 4We have a final system way as a product of Reference 37.735401.9552

4.5

For example, taking the same GenBank file used earlier:

```
>>> from Bio import SeqIO
>>> record = SeqIO.read("NC_005816.gb", "genbank")
```



```
>>> sub_record.description = "Yersinia pestis biovar Microtus str. 91001 plasmid pPCP1, partial."  
>>> print sub_record.format("genbank")  
...
```

See Sections [16.1.7](#) and [16.1.8](#)





## 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"):
```









## 5.2 Parsing sequences from the net

In the previous section, we looked at parsing sequence data from a file. Instead of using a filename, you can give `Bi o. SeqIO` a handle (see Section

```

handle = Entrez.efetch(db="nucleotide", rettype="gb", retmode="text", \
                        id="6273291,6273290,6273289")
for seq_record in SeqIO.parse(handle, "gb"):
    print seq_record.id, seq_record.description[:50] + "..."
    print "Sequence length %i," % len(seq_record),
    print "%i features," % len(seq_record.features),
    print "from: 8stures," % len(seq_rannotaescrs["source"]9):

```

- `Bio.SeqIO.to_dict()` is the most flexible but also the most memory demanding option (see Section 5.3.1)

#### 5.3.1.1 Specifying the dictionary keys

This should give:

Z78533.1 JUeWn6DPhgZ9nAyowsgtoD9TTo

Z78532.1 MN/s0q9zDoCvEEc+k/I FwCNF2pY

...

Z78439.1 H+JfaShya/4yyAj 7I bMqgNkxdxQ

Now, recall the



```

>>> from Bio import SeqIO
>>> orchid_dict = SeqIO.index("Is_orchid.fasta", "fasta")
>>> len(orchid_dict)
94
>>> orchid_dict.keys()
['gi|2765596|emb|Z78471.1|PDZ78471', 'gi|2765646|emb|Z78521.1|CCZ78521', ...
..., 'gi|2765613|emb|Z78488.1|PTZ78488', 'gi|2765583|emb|Z78458.1|PHZ78458']

```

#### 5.3.2.1 Specifying the dictionary keys





SeqRecord will write them to a FASTA format like:

```
rec3 = SeqRecord(Seq("MVTVEEFRAQCAEGPATVMAI GTATPSNCVDQSTYQI YFRI TNSEHKVELKEKFKRMC" =
```







## Chapter 6

# Multiple Sequence Alignment objects

This chapter is about Multiple Sequence Alignments, by which we mean a collection of multiple sequences which have been aligned together – usually with the insertion of gap characters, and addition of leading or trailing gaps – such that all the sequence strings are the same length. Such an alignment can be regarded as a matrix of letters, where each row is held as a SeqRecord object internally.

We will introduce the MultipleSeqAlignment object which holds this kind of data, and the Bio.AlignIO









Epsilon	CCCAAC
...	
5	6
Alpha	AAAACC
Beta	ACCCCC
Gamma	AAAACC
Delta	CCCCAA
Epsilon	CAAACC

If you wanted to read this in using `Bio.AlignIO` you could use:

```
from Bio import AlignIO
alignments = AlignIO.parse("resampled.phy", "phylip")
for alignment in alignments:
    print alignment
    print
```

This would give the following output, again abbreviated for display:

```
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAACCA Alpha
AAACCC Beta
ACCCCA Gamma
CCCAAC Delta
CCCAAA Epsilon
```

```
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAACAA Alpha
AAACCC Beta
ACCCAA Gamma
CCCACC Delta
CCCAAA Epsilon
```

```
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAAAAC Alpha
AAACCC Beta
AACAAC Gamma
CCCCCA Delta
CCCAAC Epsilon
```

...

```
SingleLetterAlphabet() alignment with 5 rows and 6 columns
AAAACC Alpha
ACCCCC Beta
AAAACC Gamma
CCCCAA Delta
CAAACC Epsilon
```

As with the function `Bio.SeqIO.parse()`, using `Bio.AlignIO.parse()` returns an iterator. If you want to keep all the alignments







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_BP I KE/1-52	RA
COATB_BP I 22/32-83	KA
COATB_BPM13/24-72	KA
COATB_BPZJ2/1-49	KA
Q9T009_BPFD/1-49	KA
COATB_BP I F1/22-73	RA



First of all, in some senses the alignment objects act like a Python list of SeqRecord objects (the rows). With this model in mind hopefully the actions of len() (the number of rows) and iteration (each row as a SeqRecord) make sense:

```
>>> from Bio import AlignIO
>>> alignment = AlignIO.read("PF05371_seed.sth", "stockholm")
>>> print "Number of rows: %i" % len(alignment)
Number of rows: 7
>>> for record in alignment:
...     print "%s - %s" % (record.seq, record.id)
AEPNAATNYATEAMDSLKTQAI DLI SQTWPVVTTVVAGLVI RLFKKFSSKA - COATB_BPI KE/30-81
AEPNAATNYATEAMDSLKTQAI DLI SQTWPVVTTVVAGLVI KLFKKFVSRA - Q9T0Q8_BPI KE/1-52
DGTSTATSYATEAMNSLKTQATDLI DQTWPVVTSTAVAGLAI RLFKKFSSKA - COATB_BPI 22/3-E57("%s)-525(-)a525(%) -525(-)----A
```



DGTSTAATEAMNSLKTQATDLI DQTWPVVTSAVAVAGLAI RLFKKFSSKA COATB\_BPI 22/32-83  
AEGDDPAKAAFNSLQASATEYI GYAWAMVVVI VGATI GI KLFKKFTSKA COATB\_BPM13/24-72  
AEGDDPAKAAFDSLQASATEYI GYAWAMVVVI VGATI GI KLFKKFASKA COATB\_BPZJ2/1-49  
AEGDDPAKAAFDSLQASATEYI GYAWAMVVVI VGATI GI KLFKKFTSKA Q9T0Q9\_BPF1/1-49  
FAADDAKAAAFDSLTAQATEMSGYAWALVVLVGATVGI KLFKKFVSRA COATB\_BPI F1/22-73

Another common use of alignment addition would be to combine alignments for several different genes

#### 6.4.1 ClustalW









```
>>> from Bio.Align.Applications import MuscleCommandline
>>> muscle_cli = MuscleCommandline(cwlstrict=True)
>>> print muscle_cli
muscle -051
```

```
>>> SeqIO.write(records, handle, "fasta")
6
>>> data = handle.getvalue()
```

You can then run the tool and parse the alignment as follows:

```
>>> stdout, stderr = muscle_cline(stdin=data)
>>> from Bio import AlignIO
>>> align = AlignIO.read(StringIO(stdout), "clustal")
>>> print align
SingleLetterAlphabet() alignment with 6 rows and 900 columns
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273290|gb|AF191664.1|AF19166
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAG...AGA gi|6273289|gb|AF191663.1|AF19166
```







After doing this, the results are in the file

This section will show briefly how to use these tools from within Python. If you have already read or tried the alignment tool examples in Section



versions of BLAST. Not only is the XML output more stable than the plain text and HTML output, it is



```

...         print 'e value:', hsp.expect
...         print hsp.query[0:75] + '...'
...         print hsp.match[0:75] + '...'
...         print hsp.subject[0:75] + '...'

```

This will print out summary reports like the following:

```

****Alignment****
sequence: >gb|AF283004.1|AF283004 Arabidopsis thaliana cold acclimation protein WCOR413-like protein
alpha form mRNA, complete cds
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. This will, of course, depend on what you want to use it for, but hopefully this helps you get started on doing what you need to do!

An important consideration for extracting information from a BLAST report is the type of objects that



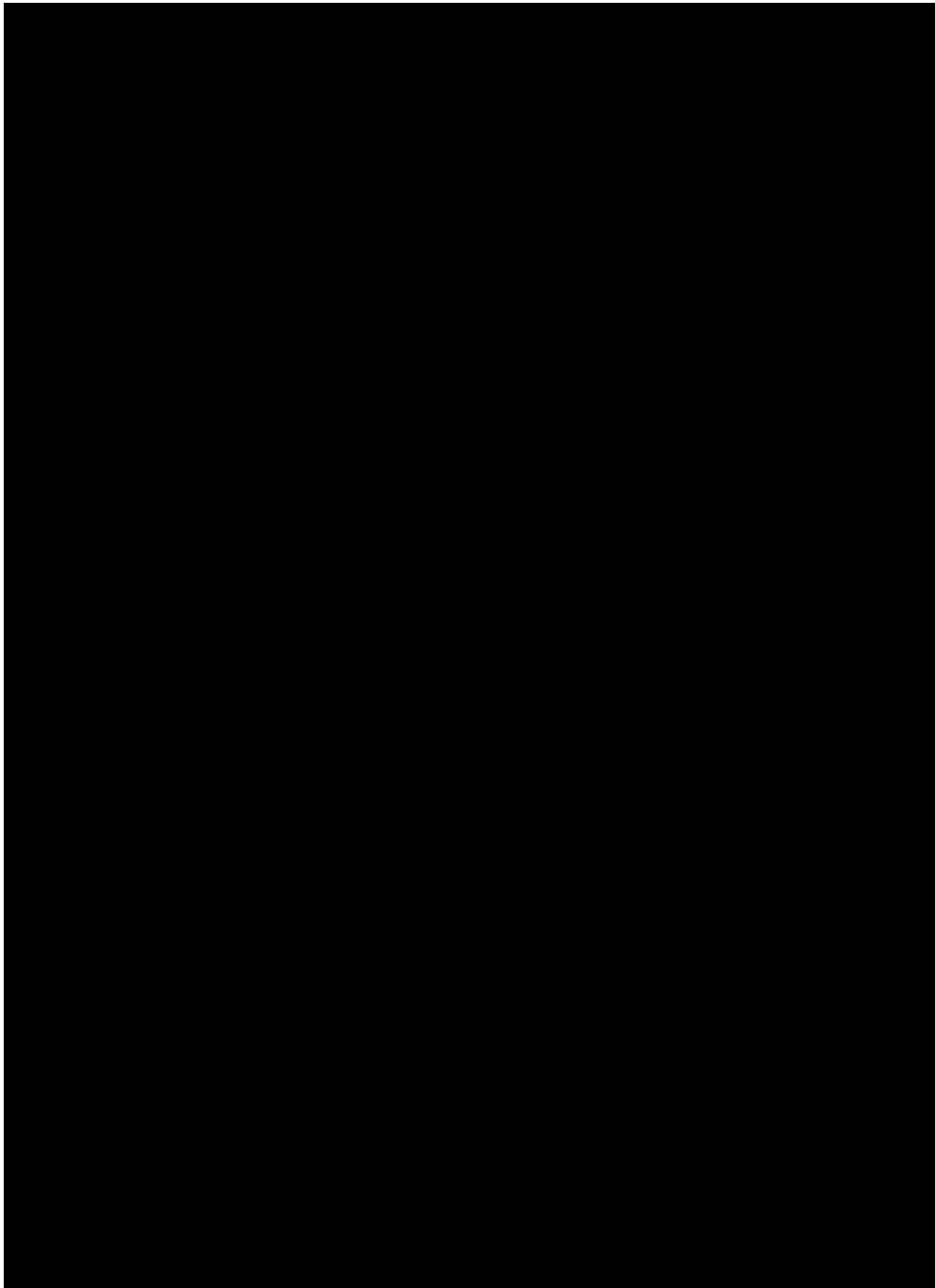


Figure 7.2: Class diagram for the PSIBlast Record class.





- `ValueError` – This is the same error generated by the regular `BlastParser`, and is due to the parser not





The Entrez Programming Utilities can also generate output in other formats, such as the Fasta or

## 8.2 EInfo: Obtaining information about the Entrez databases

```
      <DbName>uni gene</DbName>
      <DbName>uni sts</DbName>
</DbLi st>
</el nfoResul t>
```



list of IDs, the database etc, are all turned into a long URL sent to the server. If your list of IDs is long, this URL gets long, and long URLs can break (e.g. some proxies don't cope well).

Instead, you can break this up into two steps, first uploading the list of IDs using EPost (this uses an







```
Seq(' ATTTTTACGAACCTGTGGAAATTTTGGTTATGACAATAAATCTAGTTTAGTA...GAA', IUPACAmbiguousDNA())
```

Note that a more typical use would be to save the sequence data to a local file, and *then* parse it with `Bio.SeqIO`

The record variable consists of a Python list, one for each database in which we searched. Since we

## 8.8 EGQuery: Global Query - counts for search terms



```
record = handler.read(handle)
File "/usr/local/lib/python2.7/site-packages/Bio/Entrez/Parser.py", line 164, in read
    raise NotXMLError(e)
Bio.Entrez.Parser.NotXMLError: Failed to parse the XML data (syntax error: line 1, column 0). Please make
```

Here, the parser didn't find the `<?xml ...` tag with which an XML file is supposed to start, and therefore decides (correctly) that the file is not an XML file.

When your file is in the XML format but is corrupted (for example, by ending prematurely), the parser will raise a `CorruptedXMLError`. Here is an example of an XML file that ends prematurely:

```
<?xml version="1.0"?>
<!DOCTYPE elnfoResul t PUBLIC "-//NLM//DTD elnfoResul t, 11 May 2002//EN" "http://www.ncbi.nlm.nih.gov/entrez/
<elnfoResul t>
<DbLi st>
    <DbName>pubmed</DbName>
    <DbName>protei n</DbName>
    <DbName>nucl eoti de</DbName>
    <DbName>nuccore</DbName>
    <DbName>nucgss</DbName>
    <DbName>nucest</DbName>
    <DbName>structure</DbName>
    <DbName>genome</DbName>
    <DbName>books</DbName>
    <DbName>cancerchromosomes</DbName>
```



### 8.12.1 Parsing Medline records

You can find the Medline parser in `Bio.Medline`





### 8.12.2 Parsing GEO records

GEO ([Gene Expression Omnibus](#)) is a data repository of high-throughput gene expression and hybridization

PROTSIM      ORG=9986; PROTGI=126722851; PROTI D=NP\_001075655.1; PCT=76.90; ALN=288

...

PROTSIM      ORG=9598; PROTGI=114619004; PROTI D=XP\_519631.2; PCT=98.28; ALN=288

SCOUNT      38

SEQUENCE      ACC=BC067218.1; NI D=g45501306; PI D=g45501307; SEQTYPE=mRNA

SEQUENCE      ACC=NM\_000015.2; NI D=g116295259; PI D=g116295260; SEQTYPE=mRNA

SEQUENCE      ACC=D90042.1; NI D=g219415; PI D=g219416; SEQTYPE=mRNA

SEQUENCE      ACC=D90040.1; NI D=g219411; PI D=g219412; SEQTYPE=mRNA

SEQUENCE      ACC=BC015878.1; NI D=g16198419; PI D=g16198420; SEQTYPE=mRNA

SEQUENCE      ACC=CL-19638.1; NI D=711P\_5806; PI D=711P\_5912; SEQTYPE=mRNA

SEQUENCE      ACCG56929378.1; NI D=357694416; NI 698912; SEQTYPEST12;

...

SEQUENCE      ACAU09953478.1; NI D=355066316; NI 880012; SEQTYPESTNA

## 8.13 Using a proxy





```
>>> print records[0]["GBSeq_other-seqids"]  
['gb|DQ110336.1|', 'gi|187237168']
```

```
>>> print records[0]["GBSeq_definition"]  
Cypripedium calceolus voucher Davis 03-03 A maturase (matR) gene, partial cds;  
mitochondrial
```

```
>>> print records[0]["GBSeq_organism"]  
Cypripedium calceolus
```

You could use this to quickly set up searches – but for heavy usage, see Section [8.15](#).

### 8.14.3 Searching, downloading, and parsing GenBank records

```
>>> text = handle.read()
>>> print text
LOCUS      AY851612                892 bp    DNA        linear    PLN 10-APR-2007
DEFINITION Opuntia subulata rpl 16 gene, intron; chloroplast.
ACCESSION  AY851612
```

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



When you get the XML output back, it will still include the usual search results:

```
>>> gi_list = search_results["IdList"]
>>> count = int(search_results["Count"])
>>> assert count == len(gi_list)
```

```
fetch_handle = Entrez.efetch(db="pubmed",  
                             rettype="medline", retmode="text",  
                             retstart=start, retmax=batch_size,  
                             webenv=search_results["WebEnv"],  
                             query_key=search_results["QueryKey"])  
data = fetch_handle.read()  
fetch_handle.close()
```

## Chapter 9

# Swiss-Prot and ExPASy

### 9.1 Parsing Swiss-Prot files

Swiss-Prot (<http://www.expasy.org/>)

```
>>> 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.txtk41st
>>> = KeyWLi.parseseen(handtk41st)]TJ0-11.9251Td[(>>>)-52for>>
```



```
CC  -! - Also hydrolyzes diacylglycerol .
PR  PROSITE; PDOC00110;
DR  P11151, LIPL_BOVIN ; P11153, LIPL_CAVPO ; P11602, LIPL_CHICK ;
DR  P55031, LIPL_FELCA ; P06858, LIPL_HUMAN ; P11152, LIPL_MOUSE ;
DR  046647, LIPL_MUSVI ; P49060, LIPL_PAPAN ; P49923, LIPL_PIG ;
DR  Q06000, LIPL_RAT ; Q29524, LIPL_SHEEP ;
//
```

In this example, the first line shows the EC (Enzyme Commission) number of lipoprotein lipase (sec-



## 9.5 Accessing the ExPASy server



```
>>> 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

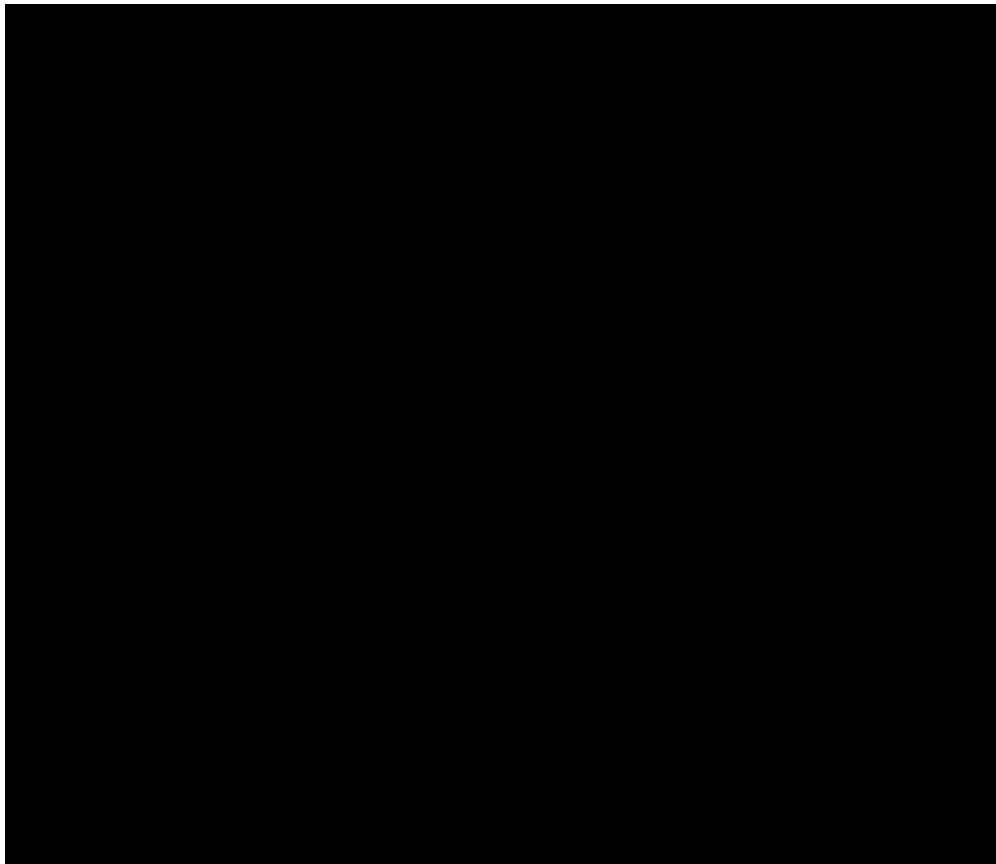


Figure 10.1: UML diagram of the SMCRA data structure used to represent a macromolecular structure.

```
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





#### 10.5.1.1 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 residues Thr A3, ..., Gly A202, Leu A3, Glu A204. Clearly, Leu A3 should be Leu A203. A couple of similar situations exist for structure 1FFK



Chapter 11

**Bio.PopGen: Population genetics**





11.2ce1t simul1

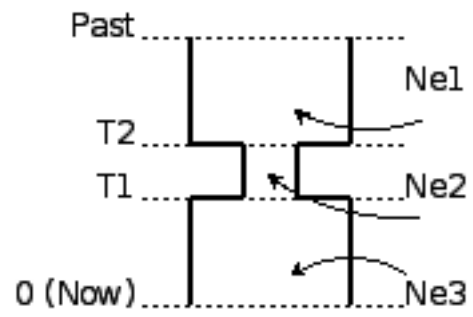


Figure 11.1: A bottleneck

#### 11.2.1.2 Chromosome structure

We strongly recommend reading SIMCOAL2 documentation to understand the full potential available in







In practice, when the number of populations is low, the mutation model is stepwise and the sample size

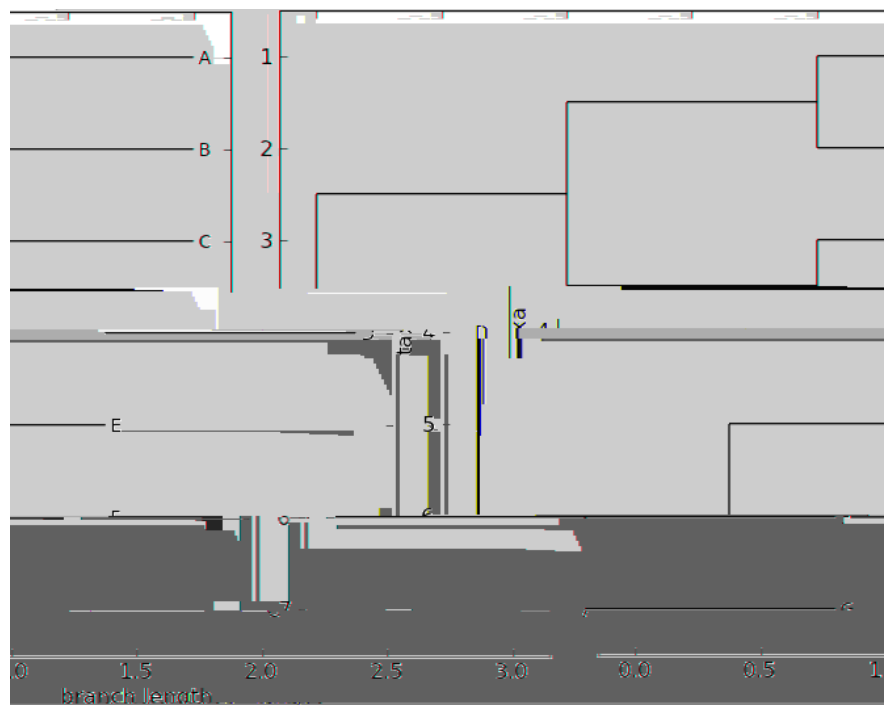
## Chapter 12

# Phylogenetics with Bio.Phylo



```
        Clade(branch_length=1.0, name="C")
        Clade(branch_length=1.0, name="D")
    Clade(branch_length=1.0)
        Clade(branch_length=1.0, name="E")
        Clade(branch_length=1.0, name="F")
        Clade(branch_length=1.0, name="G")
```

The Tree object contains global information about the tree, such as whether it's rooted n2u-336r7(r)1(o)337(tr)1d337. I







## 12.3 View and export trees

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

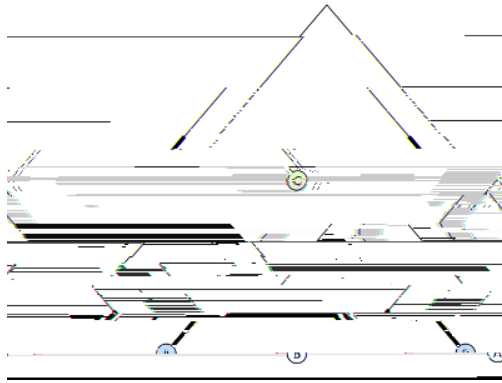
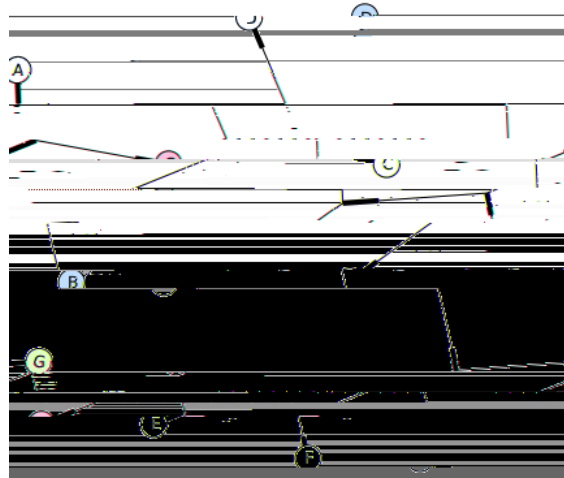


Figure 12.3: A simple rooted tree drawn with `draw_graphviz`, using `dot` for node layout.



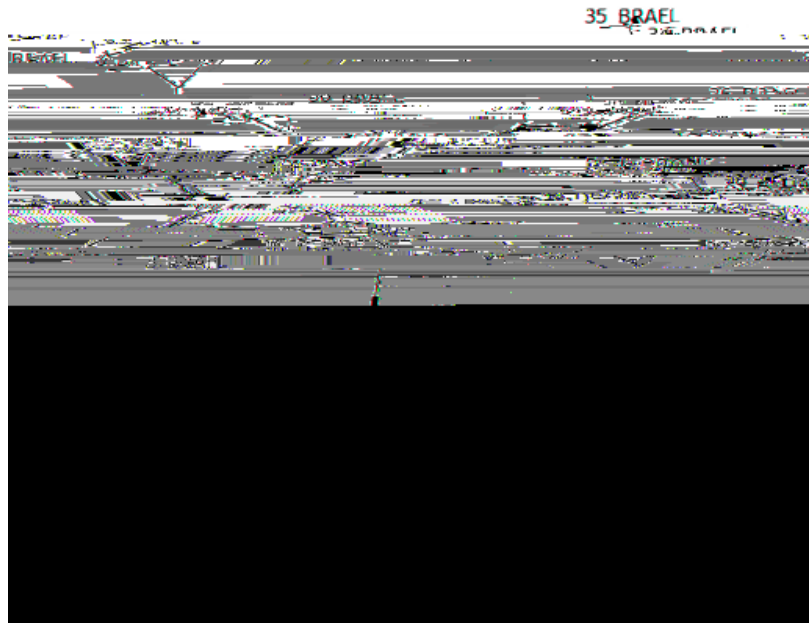


Figure 12.6: A larger tree, using







### 12.4.2 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

## 12.6 PAML integration

Biopython 1.58 brings support for PAML (<http://abacus.gene.ucl.ac.uk/software/paml.html>), a suite of programs for phylogenetic analysis by maximum likelihood. Currently the programs codeml, baseml and

Support for Newick and Nexus formats was added by porting part of the existing Bio.Nexus module to the new classes used by Bio.Phylo.

## Chapter 13

# Sequence motif analysis using Bio.Motif

This chapter gives an overview of the functionality of the Bio.Motif package included in Biopython distribution. his ineornpeesaresvvdn(in)-304(anayi)1(s)-1his of s,n sl'lln (s)-1sumef thtfarde



```
' G' : -2.3219280948873622},
{' A' : 1.7655347463629771,
' C' : -2.3219280948873622,
' T' : -2.3219280948873622,
' G' : -2.3219280948873622},
{' A' : -2.3219280948873622,
' C' : -2.3219280948873622,
' T' : 1.7655347463629771,
' G' : -2.3219280948873622},
{' A' : 1.3785116232537298,
' C' : -2.3219280948873622,
' T' : 0.0,
' G' : -2.3219280948873622},
{' A' : 1.7655347463629771,
' C' : -2.3219280948873622,
' T' : -2.3219280948873622,
' G' : -2.3219280948873622}
]
```



```
>>> arnt.make_counts_from_instances()
```

```
test_seq=Seq("TATGATGTAGTATAATATAATTATAA", m. al phabet)
```

or a threshold satisfying (roughly) the equality between the false-positive rate and the  $-\log$  of the information content (as used in patser software by Hertz and Stormo).

For example, in case of our motif, you can get the threshold giving you exactly the same results (for this sequence) as searching for instances with balanced threshold with rate of 1000.

```
>>> for pos,score in m.search_pwm(test_seq, threshold=sd.threshold_balanced(1000)):
...     print pos,score
...
10 8.44065060871
15 8.44065060871
-20 8.44065060871
21 8.44065060871
```

### 13.1.3 Comparing motifs

Once we have more than one motif, we might want to compare them. For that, we have currently three different methods of Bio.Motif objects.

bATTA  
TATAA

There are two other functions: `dist_dpq`

### 13.2.2 **AlignAce**

We can do very similar things with AlignACE program. Assume, you have your output in the file



The logistic regression model gives us appropriate values for the parameters  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$  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



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 14.1.

#### 14.2.2 Initializing a $k$

```
...
>>> 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

True: 1 Predicted: 1  
True: 1 Predicted: 0  
True: 1 Predicted: 1  
True: 1 Predicted: 1  
True: 1 Predicted: 1  
True: 1 Predicted: 1  
True: 1 Predicted: 1  
True: 1 Predicted: 1  
True: 1 Predicted: 1  
True: 1 Predicted: 0  
True: 0 Predicted: 0  
True: 0 Predicted: 0  
True: 0 Predicted: 1  
True: 0 Predicted: 0  
True: 0 Predicted: 0  
True: 0 Predicted: 0  
True: 0 Predicted: 1

The leave-one-out analysis shows that  $k$ -nearest neighbors model is correct for 13 out of 17 gene pairs, which







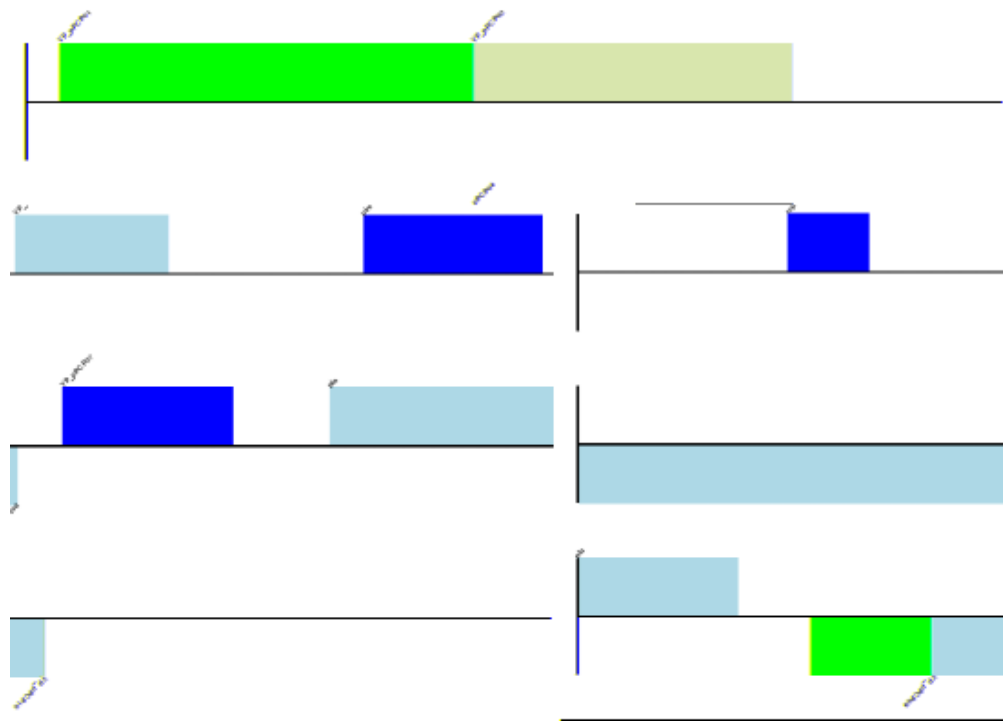


Figure 15.1: Simple linear diagram for

#### 15.1.4 A bottom up example

Now let's produce exactly the same figures, but using the bottom up approach. This means we create the different objects directly (and this can be done in almost any order) and then combine them.

```
from reportlab.lib import colors
from reportlab.lib.units import cm
from Bio.Graphics import GenomeDiagram
from Bio import SeqIO
record = SeqIO.read("NC_005816.gb", "genbank")
```

```
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,
```



### 15.1.7 Feature sigils



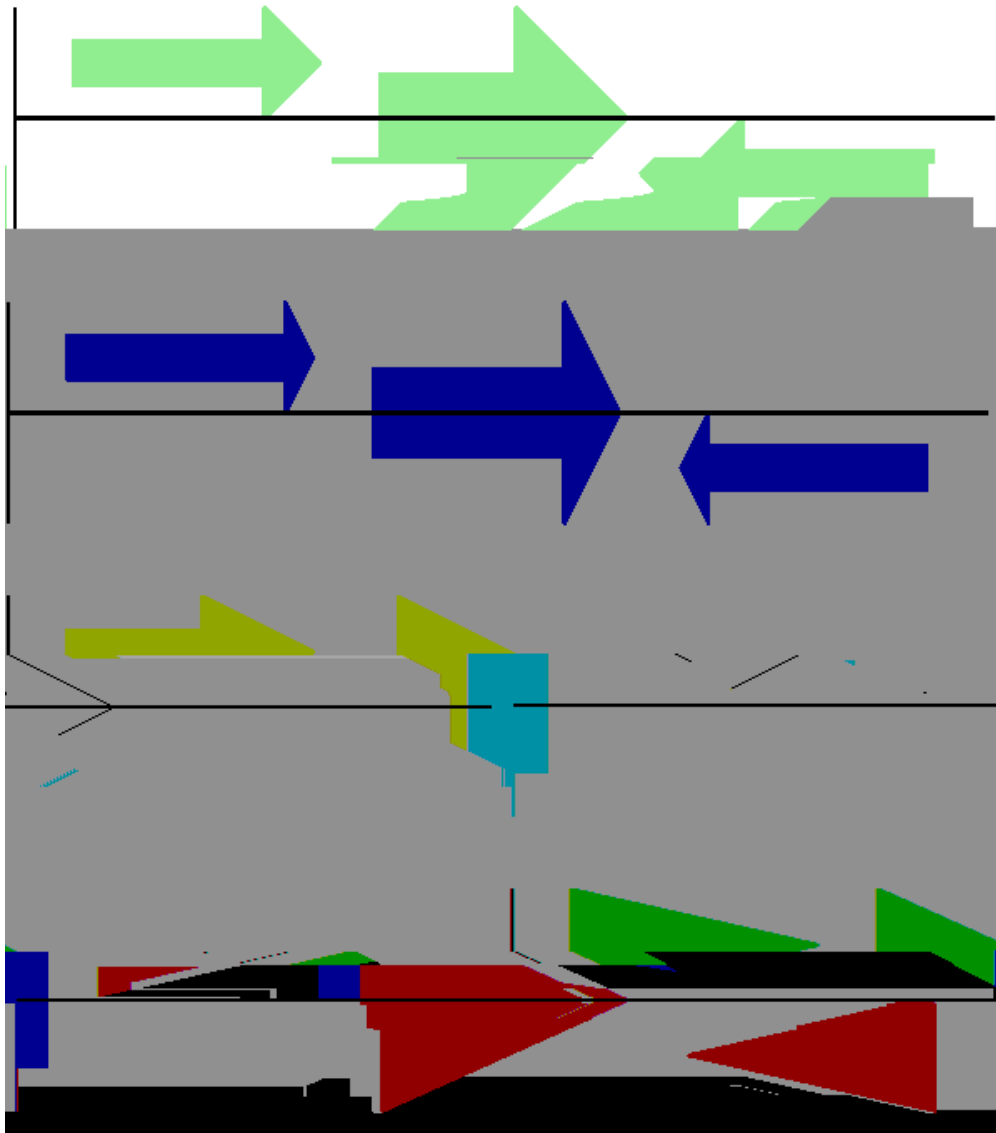


Figure 15.5: Simple GenomeDiagram showing arrow head options (see Section [15.1.7](#))

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



The expected output is shown in Figures 15.6 and 15.7.

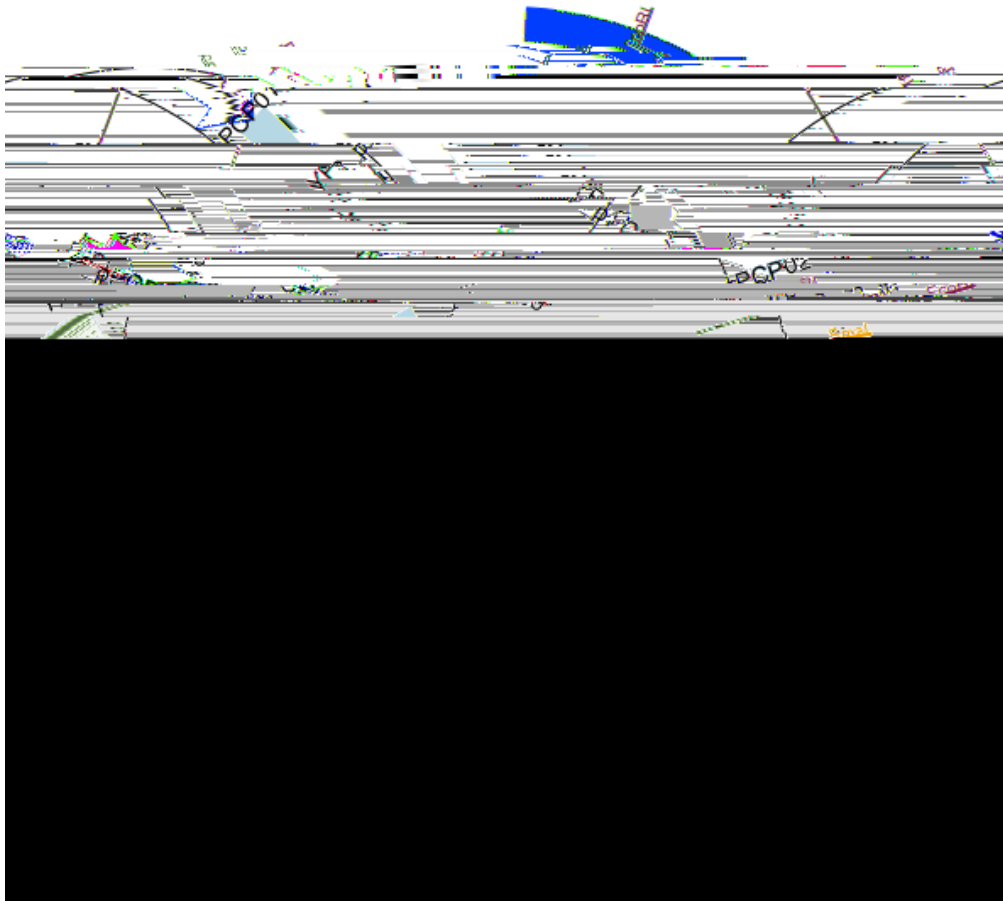


Figure 15.7: Circular diagram for *Yersinia pestis* biovar *Microtus*

```
gdd = GDDiagram("An example")  
...
```

you could just switch the import statements like this:

```
from Bio.Graphics.GenomeDiagram import FeatureSet as GDFeatureSet, Diagram as GDDiagram  
gdd = GDDiagram("An example")  
...
```

and hopefully that should be enough. In the long term you might want to switch to the new names, but you



```

entries = [("Chr I", 30432563),
           ("Chr II", 19705359),
           ("Chr III", 23470805),
           ("Chr IV", 18585042),
           ("Chr V", 26992728)]

max_len = 30432563 #Could compute this
telomere_length = 1000000 #For illustration

chr_diagram = BasicChromosome.Organism()
chr_diagram.page_size = (29.7*cm, 21*cm) #A4 landscape

for name, length in entries:
    cur_chromosome = BasicChromosome.Chromosome(name)
    #Set the scale to the MAXIMUM length plus the two telomeres in bp,
    #want the same scale used on all five chromosomes so they can be
    #compared to each other
    cur_chromosome.scale_num = max_len + 2 * telomere_length

    #Add an opening telomere
    start = BasicChromosome.TelomereSegment()
    start.scale = telomere_length
    cur_chromosome.add(start)

    #Add a body - using bp as the scale p,

```





### 16.1.2 Producing randomised genomes

Let's suppose you are looking at genome sequence, hunting for some sequence feature – maybe extreme local



Personally I prefer the following version using a function to shuffle the record and a generator expression instead of the for loop:

```
import random
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO

def make_shuffle_record(record, new_id):
    nuc_list = list(record.seq)
    random.shuffle(nuc_list)
    return SeqRecord(Seq("".join(nuc_list), record.seq.alphabet), \
        id=new_id, description="Based on %s" % original_rec.id)

original_rec = SeqIO.read("NC_005816.gb", "genbank")
```

#### 16.1.4 Making the sequences in a FASTA file upper case

Often you'll get data from collaborators as FASTA files, and sometimes the sequences can be in a mixture of upper and lower case. In some cases this is deliberate (e.g. lower case for poor quality regions), but usually

First we scan through the file once using `Bio.SeqIO.parse()`, recording the record identifiers and their

This pulled out only 14580 reads out of the 41892 present. A more sensible thing to do would be to quality trim the reads, but this is intended as an example only.

FASTQ files can contain millions of entries, so it is best to avoid loading them all into memory at once. This example uses a generator expression, which means only one

This takes longer, as this time the output file contains all 41892 reads. Again, we're used a generator

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

Because we are using a FASTQ input file in this example, the SeqRecord



#### 16.1.10 Converting FASTA and QUAL files into FASTQ files

FASTQ files hold *both* sequences and their quality string. They hold



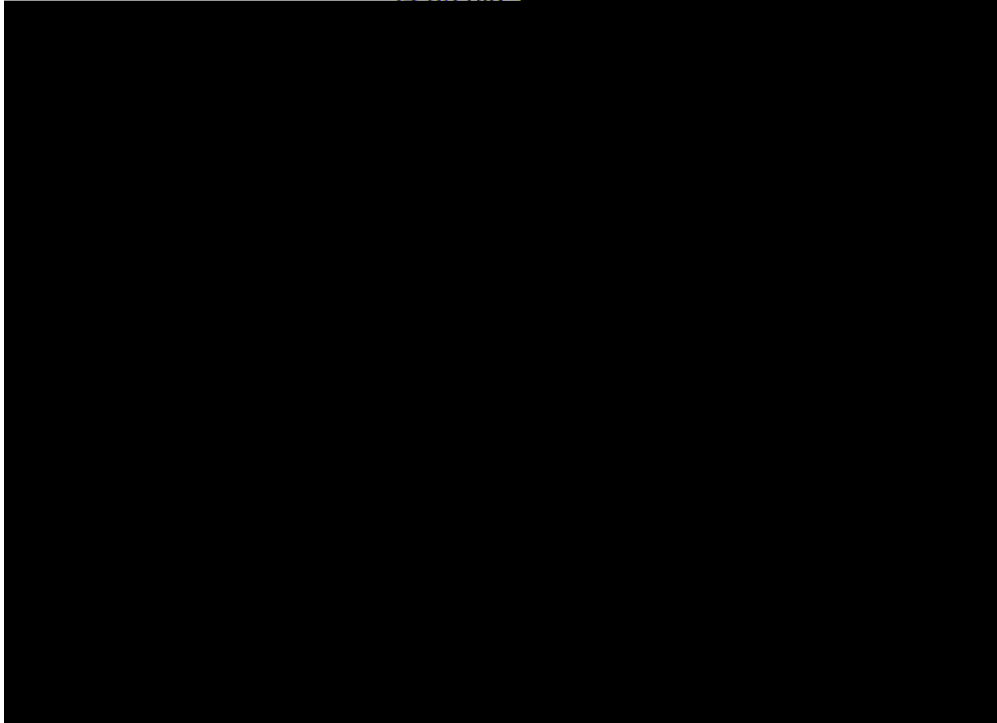


### 16.1.13 Identifying open reading frames



If however all you want to find are the locations of the open reading frames, then it is a waste of time

94 orchid sequences











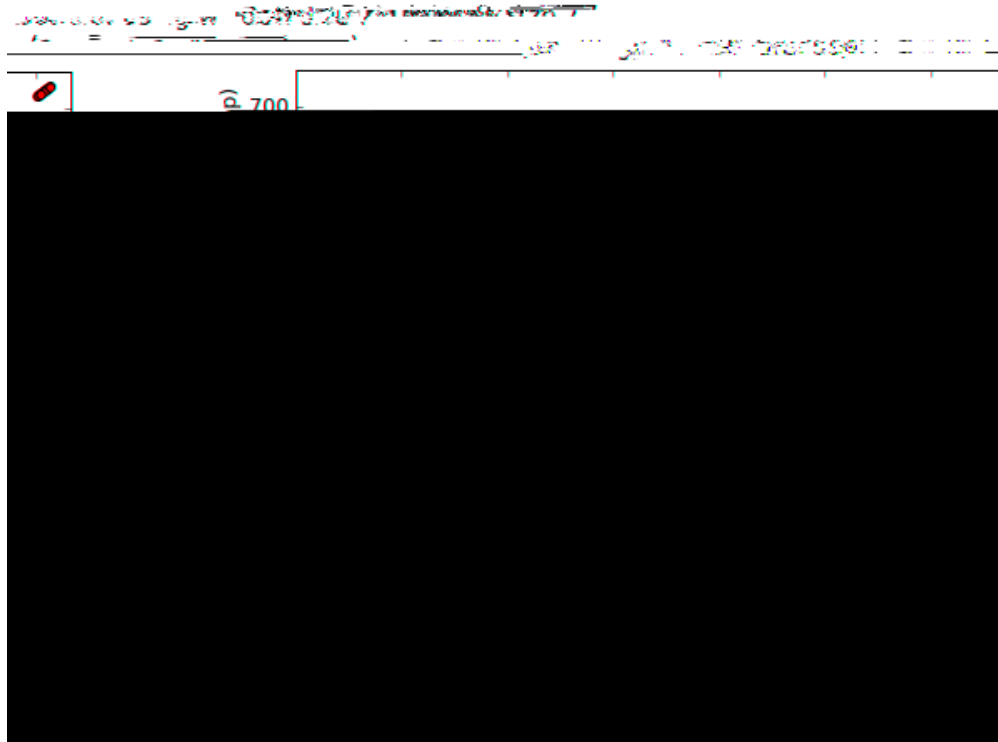


Figure 16.4: Nucleotide dot plot of two orchid sequence lengths (using pylab's scatter function).

```
import pylab
from Bio import SeqIO
for subfigure in [1,2]:
    filename = "SRR001666_%i.fastq" % subfigure
    pylab.subplot(1, 2, subfigure)
    for i,record in enumerate(SeqIO.parse(filename, "fastq")):
        if i >= 50 : break #trick!
        pylab.plot(record.letter_annotations["phred_quality"])
    pylab.ylim(0, 45)
    pylab.ylabel ("PHRED quality score")
    pylab.xlabel ("Position")
pylab.savefig("SRR001666.png")
print "Done"
```

You should note that we are using the Bio.SeqIO format name fastq here because the NCBI has saved

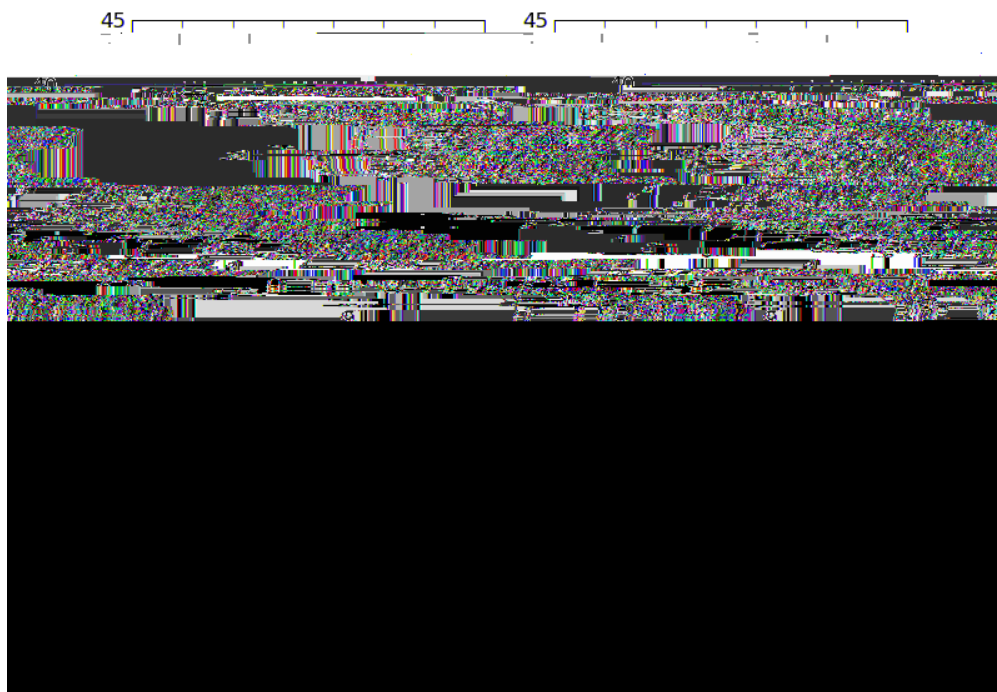


Figure 16.5: Quality plot for some paired end reads.

### 16.3.1 Calculating summary information

Once you have an alignment, you are very likely going to want to find out information about it. Instead of

```
consensus = summary_align.dumb_consensus()
```

As the name suggests, this is a really simple consensus calculator, and will just add up all of the residues

1. To maintain strictness with the alphabets, you can only include characters along the top of the PSSM that are in the alphabet of the alignment object. Gaps are not included along the top axis of the PSSM.
2. The sequence passed to be displayed along the left side of the axis does not need to be the consensus.



## 16.4 Substitution Matrices

Substitution matrices are an extremely important part of everyday bioinformatics work. They provide the



## Chapter 17







from Bio imposm





## 17.3 Writing doctests

Chapter 18

**Advanced**









Summing up to 1.

When passing a dictionary as an argument, `yyy1Fdyndi(y)1(icate7(as)-wd[(Wh)1(etd[(Wh)1(ery),)-3ity),asicod[(S)-`







Chapter 20

Appendix: Useful stuff about Python

On older versions of Biopython you had to use a handle, e.g.

```
from Bio import SeqIO
handle = open("m_cold.fasta", "r")
```