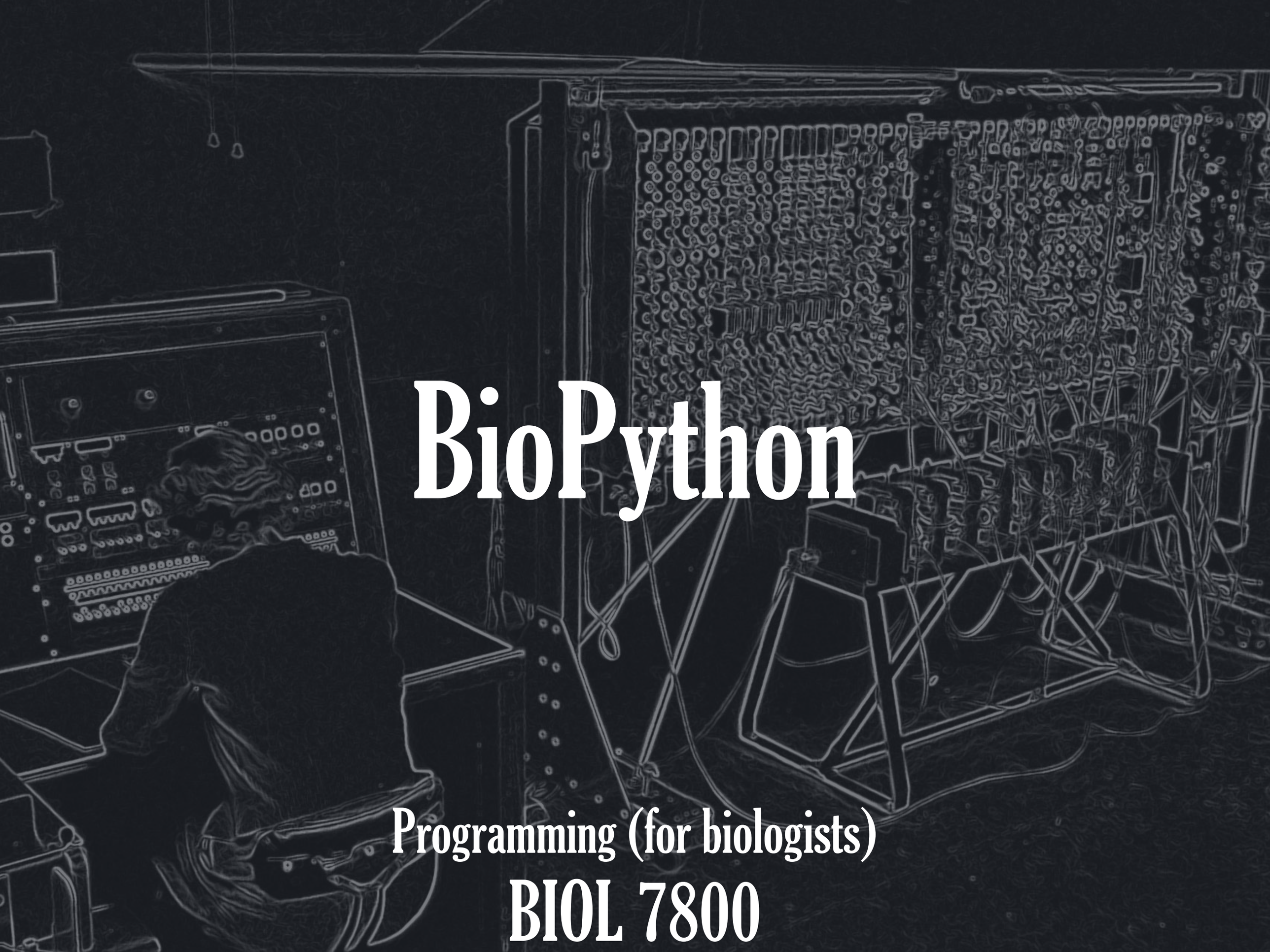

BioPython

Programming (for biologists)

BIOL 7800

Bio.SeqIO

SeqRecord objects

matts_file.fasta

```
>gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved
element locus uce-505 genomic sequence
AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGAGAAGCCAATCCCTTGA
```

```
In:  from Bio import SeqIO
```

```
In:  with open('matts_file.fasta', 'r') as infile:
         matts_seq = SeqIO.read(infile, 'fasta')
```

```
In:   matts_seq.id
Out:'gi|927657366|gb|KT692534.1|'
```

```
In:   matts_seq.name
Out:'gi|927657366|gb|KT692534.1|'
```

```
In:   matts_seq.description
Out: 'gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved element locus uce-505 genomic sequence'
```

Bio.SeqIO

SeqRecord objects

matts_file.fasta

```
>gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved
element locus uce-505 genomic sequence
AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGAGAAGCCAATCCCTTGA
```

```
In:  from Bio import SeqIO

In:  with open('matts_file.fasta', 'r') as infile:
         matts_seq = SeqIO.read(infile, 'fasta')

In:   matts_seq.id
Out:'gi|927657366|gb|KT692534.1|'

In:   matts_seq.name
Out:'gi|927657366|gb|KT692534.1|'

In:   matts_seq.description
Out 'gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved element locus uce-505 genomic
sequence'
```

id is header up to 1st space

name is often same

description is **full header**

Bio.SeqIO

SeqRecord objects have a format() method

matts_file.fasta

```
>gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:4237 ultra conserved
element locus uce-505 genomic sequence
AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGAGAAGCCAATCCCTTGA
```

```
In:  from Bio import SeqIO
```

```
In:  with open('matts_file.fasta', 'r') as infile:
         matts_seq = SeqIO.read(infile, 'fasta')
```

```
In:   matts_seq.format('fasta')
```

format() takes desired format as argument

```
Out:">gi|927657366|gb|KT692534.1| Variabilichromis moorii voucher Matthew D. McGee:
4237 ultra conserved element locus uce-505 genomic
sequence\nAACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGAGAAGCC
\nAATCCCTTGA\n'
```

Bio.SeqIO

SeqIO can process a number of formats

my_file.fastq

```
@NS500216:341:HMVLYBGXX:1:11101:18457:1155 1:N:0:GTCCTTCT+CACTAGCT
CTGCAGAGAGACAAACAGACACACATTTCATTGACAAACAAAAACAGGAGACAGAGAGACATGAACAAAGAAT
+
A/AAAE//EEEEEEEEEEAEEEEEE/AEE/EEAEEEEEEEEEEEEEEAEE<EE/EEAEAEE//EE/EEE//EE
@NS500216:341:HMVLYBGXX:1:11101:17357:1167 1:N:0:GTCCTTCT+CACTAGCT
CACAAATAAGACTTATTCTGATATCCCTGAGTGTATAATCACTTGAATTTTAGTATTGTACCTGCCATAGAAC
+
AAAAAEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
```

```
In:  from Bio import SeqIO
In:  with open('my_file.fastq', 'r') as infile:
         my_seq = SeqIO.parse(infile, 'fastq')
         fastq = next(my_seq)

In:  fastq.id
Out:'NS500216:341:HMVLYBGXX:1:11101:18457:1155'

In:  fastq.letter_annotations
Out:{'phred_quality': [32,
 14,
 32,
 32,
 32
...
```

← FASTQ here, not FASTA

← Qualities, converted to PHRED

Bio.SeqIO

SeqIO can also write SeqRecords

and **only** SeqRecords!!

```
In:  from Bio import Seq
In:  from Bio.Alphabet import IUPAC
In:  from Bio import SeqRecord
In:  from Bio import SeqIO

In:  seq_string = 'ACGTACGT'
In:  seq_object = Seq.Seq(seq_string, IUPAC.ambiguous_dna)
```

Create **sequence object** from string

```
In:  seq_record_object = SeqRecord.SeqRecord(
                              seq_object,
                              id="my_first_seq",
                              name=""
                              description=""
                              )
```

Create **sequence record** from sequence object

```
In:  with open('my_file.fasta', 'w') as outfile:
         SeqIO.write([seq_record_object], outfile, 'fasta')
```

Write out a **list** of all **sequence records** to a file

Needs to be a list of SequenceObjects to write

Bio.SeqIO

But what if you don't want to make a huge list of all your sequences to write to an outfile?

```
In:  all_my_seqs = []
In:  with open('giant_fastq_file.fastq', 'r') as monstrous_infile:
         for record in SeqIO.parse(monstrous_infile, 'fastq'):
             all_my_seqs.append(record)
In:  all_my_seqs
Out:[SeqRecord_1, SeqRecord_2, ..., SeqRecord_30000000]
In:  len(all_my_seqs)
Out: 30000000
In:  with open('my_absurdly_huge_file.fasta', 'w') as outfile:
         SeqIO.write([seq_record_object], outfile, 'fasta')
```

How could you do this differently (or iteratively)?

Bio.SeqIO

But what if you don't want to make a huge list of all your sequences to write to an outfile?

How could you do this differently (or iteratively)?

```
In:  all_my_seqs = []
In:  with open('giant_fastq_file.fastq', 'r') as monstrous_infile:
         with open('my_absurdly_huge_file.fasta', 'w') as huge_outfile:
             for record in SeqIO.parse(monstrous_infile, 'fastq'):
                 huge_outfile.write(record.format('fasta'))
```

Use the `SeqRecord` `format()` method!

Bio.AlignIO

The BioPython submodule for reading/writing alignment data

```
#NEXUS
begin data;
        dimensions ntax=31 nchar=53;
        format datatype=dna missing=? gap=-;
matrix
aimophila_rufescens        gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
ammodramus_leconteii       gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
ammodramus_savannarum      gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
amphispiza_belli           gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
amphispiza_bilineata       gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
arremon_aurantiirostris    gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
arremon_brunneinucha       gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
arremonops_rufivirgatus    --ccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
atlapetes_citrinellus      gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
atlapetes_gutturalis       gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
calamospiza_melanocorys    gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
chlorospingus_ophthalmicus gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
chondestes_grammacus       gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
geoFor1                    gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
geothlypis_trichas         gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
icterus_gularis            gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
junco_phaeonotus           gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
melospiza_melodia          gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
melozone_aberti            gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
melozone_leucotis          gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
oriturus_superciliosus     gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
passerculus_sandwichensis  gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
passerella_iliaca          -------gtaaactgcaggggggcagcgagagctcagagagcagcagctttaa
peucaea_ruficauda          gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
pezopetes_capitalis        gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
pipilo_chlorurus           gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
pooecetes_gramineus        -----------------------------------------------gctttaa
rhynchospiza_strigiceps    gtccatagtaaactgcaggggggcagagagagcccagagagcagcagctttaa
spizella_arborea           gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
spizella_atrogularis       -------------------------------------------------------
zonotrichia_atricapilla    -----------------------------------------------gctttaa
;
end;
```

sparrow.nexus

← Sparrow alignment NEXUS format

(but not this ↑ sparrow)

Image from wikipedia

Bio.AlignIO

The BioPython submodule for reading/writing alignment data

```
>aimophila_rufescens
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>ammodramus_leconteii
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>ammodramus_savannarum
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>amphispiza_belli
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>amphispiza_bilineata
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>arremon_aurantiirostris
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>arremon_brunneinucha
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>arremonops_rufivirgatus
--ccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>atlapetes_citrinellus
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>atlapetes_gutturalis
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>calamospiza_melanocorys
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>chlorospingus_ophthalmicus
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>chondestes_grammacus
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>geoFor1
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>geothlypis_trichas
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>icterus_gularis
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa
>junco_phaeonotus
gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa

(…truncated…)
```

sparrow.fasta

← Sparrow alignment FASTA format

(but not this ↑ sparrow)

Image from wikipedia

Bio.AlignIO

The BioPython submodule for reading/writing alignment data

```
 31 53
aimophila_rufescens         gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
ammodramus_leconteii        gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
ammodramus_savannarum       gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
amphispiza_belli            gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
amphispiza_bilineata        gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
arremon_aurantiirostris     gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
arremon_brunneinucha        gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
arremonops_rufivirgatus     --ccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
atlapetes_citrinellus       gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
atlapetes_gutturalis        gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
calamospiza_melanocorys     gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
chlorospingus_ophthalmicus  gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
chondestes_grammacus        gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
geoFor1                     gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
geothlypis_trichas          gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
icterus_gularis             gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
junco_phaeonotus            gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
melospiza_melodia           gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
melozone_aberti             gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
melozone_leucotis           gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
oriturus_superciliosus      gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
passerculus_sandwichensis   gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
passerella_iliaca           -------gta aactgcaggg gggcagcgag agctcagaga gcagcagctttaa
peucaea_ruficauda           gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
pezopetes_capitalis         gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
pipilo_chlorurus            gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
pooecetes_gramineus         ---------- ---------- ---------- ---------- ------gctttaa
rhynchospiza_strigiceps     gtccatagta aactgcaggg gggcagagag agcccagaga gcagcagctttaa
spizella_arborea            gtccatagta aactgcaggg gggcagagag agctcagaga gcagcagctttaa
spizella_atrogularis        ---------- ---------- ---------- ---------- ----------
zonotrichia_atricapilla     ---------- ---------- ---------- ---------- ------gctttaa
```

← Sparrow alignment PHYLIP format

(but not this ↑ sparrow)

sparrow.phylip

Image from wikipedia

Bio.AlignIO

AlignIO gives us ways to read almost all alignment formats

NEXUS

```
In:  from Bio import AlignIO
In:  aln_nexus = AlignIO.read('sparrow.nexus', 'nexus')
In:  aln_nexus
Out:<<class 'Bio.Align.MultipleSeqAlignment'> instance (31 records of length 53, IUPACAmbiguousDNA()) at 104405860>
```

FASTA

```
In:  from Bio import AlignIO
In:  aln_fasta = AlignIO.read('sparrow.fasta', 'fasta')
In:  aln_fasta
Out:<<class 'Bio.Align.MultipleSeqAlignment'> instance (31 records of length 53, SingleLetterAlphabet()) at 104490fd0>
```

PHYLIP

use this to get "full" species names

```
In:  from Bio import AlignIO
In:  aln_phylip = AlignIO.read('sparrow.phylip', 'phylip-relaxed')
In:  aln_phylip
Out:<<class 'Bio.Align.MultipleSeqAlignment'> instance (31 records of length 53, SingleLetterAlphabet()) at 1044ad4e0>
```

Bio.AlignIO

Once read, AlignIO "equalizes" alignment formats
so there are no differences when manipulating them and
we get a Bio.Align.MultipleSeqAlignment object

```
In:  aln_nexus
Out:<<class 'Bio.Align.MultipleSeqAlignment'> instance (31 records of length 53, IUPACAmbiguousDNA()) at 104405860>

In:  aln_fasta
Out:<<class 'Bio.Align.MultipleSeqAlignment'> instance (31 records of length 53, SingleLetterAlphabet()) at 104490fd0>

In:  aln_phylip
Out:<<class 'Bio.Align.MultipleSeqAlignment'> instance (31 records of length 53, SingleLetterAlphabet()) at 1044ad4e0>
```

Bio.Align

Bio.Align.MultipleSeqAlignment objects are composed of SeqRecords
and like many python objects, we can iterate over them…

```
In:  from Bio import AlignIO
In:  aln_nexus = AlignIO.read('sparrow.nexus', 'nexus')
In:  for record in aln_nexus:
         print(record.__str__)
Out:
<SeqRecord(seq=Seq('gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa', IUPACAmbiguousDNA()),
id='aimophila_rufescens', name='aimophila_rufescens', description='', dbxrefs=[])>

<SeqRecord(seq=Seq('gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa', IUPACAmbiguousDNA()),
id='ammodramus_leconteii', name='ammodramus_leconteii', description='', dbxrefs=[])>

<SeqRecord(seq=Seq('gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa', IUPACAmbiguousDNA()),
id='ammodramus_savannarum', name='ammodramus_savannarum', description='', dbxrefs=[])>

<SeqRecord(seq=Seq('gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa', IUPACAmbiguousDNA()), id='amphispiza_belli',
name='amphispiza_belli', description='', dbxrefs=[])>

<SeqRecord(seq=Seq('gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa', IUPACAmbiguousDNA()),
id='amphispiza_bilineata', name='amphispiza_bilineata', description='', dbxrefs=[])>
```

Bio.Align

Bio.Align.MultipleSeqAlignment objects also have their own suite of methods

```
In:   aln = AlignIO.read('sparrow.nexus', 'nexus')
```

```
In:   aln.get_alignment_length()
Out:53
```

aln length (what does `len()` return?)

```
In:   aln.get_all_seqs()
Out:
[
SeqRecord(seq=Seq('gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa', IUPACAmbiguousDNA()), id='aimophila_rufescens',
name='aimophila_rufescens', description='', dbxrefs=[]),
 SeqRecord(seq=Seq('gtccatagtaaactgcaggggggcagagagagctcagagagcagcagctttaa', IUPACAmbiguousDNA()),
id='ammodramus_leconteii', name='ammodramus_leconteii', description='', dbxrefs=[])
]
```

return list of all sequences

```
In:   aln.append(SeqRecord)
```

add ONE sequence to aln

```
In:   aln.extend([SeqRecord, SeqRecord, SeqRecord])
```

add > 1 sequences to aln

```
In:   aln.format('stockholm')
```

convert the aln to '<format>'

Bio.Align

Bio.Align.MultipleSeqAlignment objects accept slicing parameters

```
In:   aln = AlignIO.read('sparrow.nexus', 'nexus')
```

```
In:   print(aln[0:3])
Out: IUPACAmbiguousDNA() alignment with 3 rows and 53 columns
gtccatagtaaactgcaggggggcagagagagctcagagagcag...taa aimophila_rufescens
gtccatagtaaactgcaggggggcagagagagctcagagagcag...taa ammodramus_leconteii
gtccatagtaaactgcaggggggcagagagagctcagagagcag...taa ammodramus_savannarum
```

This accesses first three **rows** of an alignment

How do we get columns?

Bio.Align

Bio.Align.MultipleSeqAlignment objects accept slicing parameters

```
In:  aln = AlignIO.read('sparrow.nexus', 'nexus')
```

```
In: print(aln[: , 0:3])
Out: IUPACAmbiguousDNA() alignment with 31 rows and 3 columns
gtc aimophila_rufescens
gtc ammodramus_leconteii
gtc ammodramus_savannarum
gtc amphispiza_belli
gtc amphispiza_bilineata
gtc arremon_aurantiirostris
gtc arremon_brunneinucha
--c arremonops_rufivirgatus
gtc atlapetes_citrinellus
gtc atlapetes_gutturalis
gtc calamospiza_melanocorys
gtc chlorospingus_ophthalmicus
gtc chondestes_grammacus
gtc geoFor1
gtc geothlypis_trichas
gtc icterus_gularis
gtc junco_phaeonotus
gtc melospiza_melodia
...
--- zonotrichia_atricapilla
```

How do we get
3 rows and 3 columns?

Bio.Align

Bio.Align.MultipleSeqAlignment objects accept slicing parameters

```
In:   aln = AlignIO.read('sparrow.nexus', 'nexus')
```

How do we get 3 rows and 3 columns?

Slice columns

Slice rows

```
In:   print(aln[0:3 , 0:3])
Out: IUPACAmbiguousDNA() alignment with 3 rows and 3 columns
gtc aimophila_rufescens
gtc ammodramus_leconteii
gtc ammodramus_savannarum
```

Bio.AlignIO

When we are done manipulating our Bio.Align.MultipleSeqAlignment objects we can use Bio.AlignIO to **write** out our alignment

```
In:  aln = AlignIO.read('sparrow.nexus', 'nexus')
```
← Read in alignment

```
In:  short_aln = aln[0:10, 0:25]
```
← Slice alignment: 10 rows, 25 cols

```
In:  with open('my_short_align.phylip', 'w') as outfile:
         AlignIO.write(short_aln, outfile, 'phylip')
```
← open output file

← write alignment as `phylip`

Bio.Restriction

A set of classes for performing *in-silico* restriction digests

The `Restriction` sub-module contains many different enzyme `classes`

```
In:   from Bio import Restriction
In:   dir(Restriction)
Out: ['AanI',
 'AarI',
 'AasI',
 'AatII',
 'AbaSI',
 'AbsI',
 'Acc16I',
(...truncated...)
]
```

Bio.Restriction

The Restriction sub-module contains many different enzyme classes

Each enzyme class has it's own methods and attributes

```
In:  from Bio import Restriction
In:  dir(Restriction.EcoRI)
Out:
```

```
['all_suppliers',
'buffers',
'catalyse',
'catalyze',
'charac',
'characteristic',
'compatible_end',
'compsite',
'cut_once',
'cut_twice',
'dna',
'elucidate',
'equischizomers',
'freq',
'frequency',
'fst3',
'fst5',
'inact_temp',
'is_3overhang',
'is_5overhang',
'is_ambiguous',
'is_blunt',
'is_comm',
'is_defined',
'is_equischizomer',
'is_isoschizomer',
'is_methylable',
'is_neoschizomer',
'is_palindromic',
'is_unknown',
'isoschizomers',
'mro',
'neoschizomers',
'opt_temp',
'overhang',
'ovhg',
'ovhgseq',
'results',
'scd3',
'scd5',
'search',
'site',
'size',
'substrat',
'suppl',
'supplier_list',
'suppliers']
```

Bio.Restriction

The Restriction sub-module contains many different enzyme classes
Each enzyme class has it's own methods and attributes
And, many of these are ***extremely helpful***

```
In:  from Bio import Restriction
In:  Restriction.EcoRI.all_suppliers()

Out: Agilent Technologies,
Bangalore Genei,
EURx Ltd.,
Life Technologies,
Minotech Biotechnology,
Molecular Biology Resources - CHIMERx,
New England Biolabs,
Nippon Gene Co., Ltd.,
Promega Corporation,
Roche Applied Science,
SibEnzyme Ltd.,
Sigma Chemical Corporation,
SinaClon BioScience Co.,
Takara Bio Inc.,
Thermo Scientific Fermentas,
Toyobo Biochemicals,
Vivantis Technologies
```

Bio.Restriction

The Restriction sub-module contains many different enzyme classes
Each enzyme class has it's own methods and attributes
And, many of these are ***extremely helpful***

```
In:  from Bio import Restriction
In:  Restriction.EcoRI.elucidate()
Out: 'G^AATT_C'

In:  Restriction.EcoRI.is_blunt()
Out: False

In:  Restriction.EcoRI.opt_temp
Out: 37

In:  Restriction.EcoRI.is_5overhang()
Out: True

In:  Restriction.EcoRI.is_3overhang()
Out: False
```

Bio.Restriction

The Restriction sub-module contains many different enzyme classes

Of course, we can use these classes to *virtually digest* a sequence

mcgee_sequence.fasta

```
>gi|927657366|gb|KT692534.1|
AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGAGAAGCCAATCCCTTGA
AACTTCTCGAATCGGGCTTTTATTTTTCTTTATCCAAGATAGCTTTCCAGTTCACTTGCAGATTGCGGCA
GCCTTTCAGACGGCACACCTGTTGCTTGAGCCTTGATTTCGCTTTTTGATCAGAGGGCCATTTGACAAAA
GCGGGAAAGAAGAGGCTGAAGAAGAGGGAGAGGCAAGAGAAAAGATCCTTATCGAATGCTCTCCACCGGA
CAATGAATTTTCCCCCACCATTCAAATGTATCTCCAGAATGCTTGAGTGATTCATTACAGCAGATTCTGA
AGTGTGACAATGGACCTCAGAAATGAGTGATGCCTAGTTTTTGCTGATTTCATTTCCCTGATCCTCCTTA
TCCCTCTGGCCTGTGTACCTCTATTAATTAGAAAGTAGGGATTTCCACTCATCAACTGATGGGTTTCATG
CTCCGCTCTGTCACAAATCTCCTGTCCACGCTATCTCTCTCCTTCCAACCCATGGCTGCCTTGGAATGTA
CATACTCCAGCTCAGGCTTCAATCTCAGCAGCAAGTCAGGGTGGCTTTGGTGATTCATAAATATTTATAA
ACCGATTAAGAACGAATAATAAAGTTATAAGACAGTGGTTGGTTACCCTTGAACAGTACAACGGATGACT
TCCTTAGAAGGGGAAATGTGATCAAACGTGCCGGAAACGGTTCTTTCTCGCTATTTTGAGTGCAGAAACC
ATGTGTTTGTTATTACTCACTCGAGCGACCTCATAAAGCAATTAGCAATGTGATATTT
```

```
In:  from Bio import SeqIO

In:  with open('mcgee_sequence.fasta', 'r') as infile:
         record = SeqIO.read(infile, 'fasta')
In:   print(record)
Out: ID: gi|927657366|gb|KT692534.1|
Name: gi|927657366|gb|KT692534.1|
Description: gi|927657366|gb|KT692534.1|
Number of features: 0
Seq('AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGA...TTT',
SingleLetterAlphabet())
```

Bio.Restriction

A set of classes for performing *in-silico* restriction digests

```
In:  # record is our sequence from mcgee_sequence.fasta
In:  Restriction.EcoRI.search(record.seq)
Out: []                                          <-- EcoRI does not cut

In:  Restriction.XspI.search(record.seq)
Out: [385]                                       <-- XspI does cut
In:  Restriction.XspI.site
Out:'CTAG'                                       <-- Recognition site is CTAG
In:  Restriction.XspI.elucidate()
Out:'C^TA_G'                                     <-- ^ is cut on 5' strand; _ is cut on 3'

In:  print(record[384:])
Out:
ID: gi|927657366|gb|KT692534.1|
Name: gi|927657366|gb|KT692534.1|
Description: gi|927657366|gb|KT692534.1|
Number of features: 0
Seq('TAGTTTTTGCTGATTTCATTTCCCTGATCCTCCTTATCCCTCTGGCCTGTGTAC...TTT',
SingleLetterAlphabet())
```

Bio.Restriction

A set of classes for performing *in-silico* restriction digests
.catalyse() method of an enzyme class makes this easier

```
In:  # record is our sequence from mcgee_sequence.fasta
In:  Restriction.EcoRI.search(record.seq)
Out: []
```

```
In:  Restriction.XspI.search(record.seq)
In:  # this is cumbersome and error prone
In:  # print(record[384:])

In:  Restriction.XspI.catalyse(record.seq)
(Seq('AACTCCAGCTCCTGCAGCATCTCGCCCCATCATGACTTTTTCGGCTGATCCCGA...GCC',
SingleLetterAlphabet()),
 Seq('TAGTTTTTGCTGATTTCATTTCCCTGATCCTCCTTATCCCTCTGGCCTGTGTAC...TTT',
SingleLetterAlphabet()))
```

Returns a tuple of digested pieces

Bio.Restriction

A set of classes for performing *in-silico* restriction digests

What if you want to digest a sequence with 2 enzymes?

```
In:   # record is our sequence from mcgee_sequence.fasta
In:   my_batch = Restriction.RestrictionBatch(['XspI', 'SspI'])
In:   my_batch.search(record.seq)
Out: {XspI: [385], SspI: [623]}
```

```
In:   my_batch.catalyse(record.seq)
* * * AttributeError: 'RestrictionBatch' object has no attribute 'catalyse'
```

But there is no catalyse() method

Bio.Restriction

A set of classes for performing *in-silico* restriction digests

You can also restrict the restriction batch by supplier

```
In:  # record is our sequence from mcgee_sequence.fasta
In:  Restriction.RestrictionBatch.show_codes()
Out:
F = Thermo Scientific Fermentas
R = Promega Corporation
M = Roche Applied Science
B = Life Technologies
N = New England Biolabs
Y = SinaClon BioScience Co.
E = Agilent Technologies
U = Bangalore Genei
X = EURx Ltd.
I = SibEnzyme Ltd.
Q = Molecular Biology Resources - CHIMERx
V = Vivantis Technologies
K = Takara Bio Inc.
S = Sigma Chemical Corporation
C = Minotech Biotechnology
J = Nippon Gene Co., Ltd.
O = Toyobo Biochemicals
```

Bio.Restriction

A set of classes for performing *in-silico* restriction digests

You can also restrict the restriction batch by supplier

```
In:  # record is our sequence from mcgee_sequence.fasta
In:  # N = New England Biolabs
In:  my_batch = Restriction.RestrictionBatch(first=[], suppliers="N")
In:  my_batch
```

```
Out: RestrictionBatch(['AatII', 'AbaSI', 'Acc65I', 'AccI', 'AciI', 'AclI', 'AcuI', 'AfeI', 'AflII', 'AflIII', 'AgeI', 'AhdI', 'AleI', 'AluI', 'AlwI',
'AlwNI', 'ApaI', 'ApaLI', 'ApeKI', 'ApoI', 'AscI', 'AseI', 'AsiSI', 'AvaI', 'AvaII', 'AvrII', 'BaeGI', 'BaeI', 'BamHI', 'BanI', 'BanII', 'BbsI', 'BbvCI',
'BbvI', 'BccI', 'BceAI', 'BcgI', 'BciVI', 'BclI', 'BcoDI', 'BfaI', 'BfuAI', 'BfuCI', 'BglI', 'BglII', 'BlpI', 'BmgBI', 'BmrI', 'BmtI', 'BpmI', 'Bpu10I',
'BpuEI', 'BsaAI', 'BsaBI', 'BsaHI', 'BsaI', 'BsaJI', 'BsaWI', 'BsaXI', 'BseRI', 'BseYI', 'BsgI', 'BsiEI', 'BsiHKAI', 'BsiWI', 'BslI', 'BsmAI', 'BsmBI',
'BsmFI', 'BsmI', 'BsoBI', 'Bsp1286I', 'BspCNI', 'BspDI', 'BspEI', 'BspHI', 'BspMI', 'BspQI', 'BsrBI', 'BsrDI', 'BsrFI', 'BsrGI', 'BsrI', 'BssHII',
'BssKI', 'BssSI', 'BstAPI', 'BstBI', 'BstEII', 'BstNI', 'BstUI', 'BstXI', 'BstYI', 'BstZ17I', 'Bsu36I', 'BtgI', 'BtgZI', 'BtsCI', 'BtsI', 'BtsIMutI',
'Cac8I', 'ClaI', 'CspCI', 'CviAII', 'CviKI_1', 'CviQI', 'DdeI', 'DpnI', 'DpnII', 'DraI', 'DraIII', 'DrdI', 'EaeI', 'EagI', 'EarI', 'EciI', 'Eco53kI', 'EcoNI',
'Eco0109I', 'EcoRI', 'EcoRV', 'FatI', 'FauI', 'Fnu4HI', 'FokI', 'FseI', 'FspEI', 'FspI', 'HaeII', 'HaeIII', 'HgaI', 'HhaI', 'HinP1I', 'HincII', 'HindIII',
'HinfI', 'HpaI', 'HpaII', 'HphI', 'Hpy166II', 'Hpy188I', 'Hpy188III', 'Hpy99I', 'HpyAV', 'HpyCH4III', 'HpyCH4IV', 'HpyCH4V', 'KasI', 'KpnI',
'LpnPI', 'MboI', 'MboII', 'MfeI', 'MluCI', 'MluI', 'MlyI', 'MmeI', 'MnlI', 'MscI', 'MseI', 'MslI', 'MspA1I', 'MspI', 'MspJI', 'MwoI', 'NaeI', 'NarI',
'NciI', 'NcoI', 'NdeI', 'NgoMIV', 'NheI', 'NlaIII', 'NlaIV', 'NmeAIII', 'NotI', 'NruI', 'NsiI', 'NspI', 'PacI', 'PaeR7I', 'PciI', 'PflFI', 'PflMI', 'PleI',
'PluTI', 'PmeI', 'PmlI', 'PpuMI', 'PshAI', 'PsiI', 'PspGI', 'PspOMI', 'PspXI', 'PstI', 'PvuI', 'PvuII', 'RsaI', 'RsrII', 'SacI', 'SacII', 'SalI', 'SapI',
'Sau3AI', 'Sau96I', 'SbfI', 'ScaI', 'ScrFI', 'SexAI', 'SfaNI', 'SfcI', 'SfiI', 'SfoI', 'SgrAI', 'SmaI', 'SmlI', 'SnaBI', 'SpeI', 'SphI', 'SspI', 'StuI',
'StyD4I', 'StyI', 'SwaI', 'TaqI', 'TfiI', 'TseI', 'Tsp45I', 'TspMI', 'TspRI', 'Tth111I', 'XbaI', 'XcmI', 'XhoI', 'XmaI', 'XmnI', 'ZraI'])
```

Bio.Restriction

You can also restrict the restriction batch by supplier

And, like many other things in python, you can iterate over the batch

```
In:  # record is our sequence from mcgee_sequence.fasta
In:  # N = New England Biolabs
In:  my_batch = Restriction.RestrictionBatch(first=[], suppliers="N")
In:  for enzyme in my_batch:
         print('{!s: <10}{!s: <10}'.format(enzyme, enzyme.site))
Out:
BbsI      GAAGAC
HinfI     GANTC
TseI      GCWGC
HaeII     RGCGCY
BanI      GGYRCC
AciI      CCGC
BccI      CCATC
SphI      GCATGC
Tsp45I    GTSAC
BtsCI     GGATG
Sau3AI    GATC
MluCI     AATT
KasI      GGCGCC
(…truncated…)
```