COMP364: Biopython

Jérôme Waldispühl
McGill University

What is Biopython?

A package to make your life (for bioinformatics applications) easy!

- Parse bioinformatics files (FASTA, GenBank, PDB, etc.) and store them in appropriate data structures.
- Code to deal with popular on-line bioinformatics destinations (E.g. Blast & PubMed at NCBI).
- Interfaces to common bioinformatics programs (E.g. ClustalW, EMBOSS).
- Tools for performing common operations on sequences.
- Code to perform classification.
- Code for dealing with alignments.
- GUI-based programs to do basic sequence manipulations, translations, BLASTing, etc.
- And much more!

Starting with Biopython

Import Module:

```
>>> import Bio
```

Create a sequence object:

```
>>> import Bio.Seq
>>> s = Bio.Seq.Seq("ACGT")
>>> s
Seq('ACGT', Alphabet())
>>> print s
ACGT
```

Alphabet() defines the alphabet used by your sequences.

Sequence object

Works like strings:

```
>>> for index, letter in enumerate(s):
...     print index, letter
0 A
1 C
2 G
3 T
```

With additional capabilities:

```
>>> s.complement()
Seq('TGCA', Alphabet())
>>> s.reverse_complement()
Seq('ACGT', Alphabet())
```

Parsing (FASTA)

FASTA format:

```
>gi|2765658|emb|Z78533.1|CIZ78533
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAAT
AAACGATCGAGTGAATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCC
...
```

Read and display each entry:

```
from Bio import SeqIO
for seq_record in SeqIO.parse("input.fasta", "fasta"):
    print seq_record.id
    print repr(seq_record.seq)
    print len(seq_record)
```

```
gi|2765564|emb|Z78439.1|PBZ78439
Seq('CGTAACAAGGTTTCCGTAGGTGAA...CGC', SingleLetterAlphabet())
740
...
gi|2765564|emb|Z78439.1|PBZ78439
Seq('CATTGTTGAGATCACATAATAATT...GCC', SingleLetterAlphabet())
592
```

Parsing other formats

Biopython supports many formats: clustal, embl, genbank, phd, phylip, swiss, stockholm…

To parse them, you just need to change the 2nd argument:

```
>>> x = SeqIO.parse("input.gbk", "genbank")
```

The rest works exactly the same!

Slicing

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC",
                 IUPAC.unambiguous_dna)
```

Slice with start & stop:

```
>>> my_seq[4:12]
Seq('GATGGGCC', IUPACUnambiguousDNA())
```

Stride with step size:

```
>>> my_seq[1::3]
Seq('AGGCATGCATC', IUPACUnambiguousDNA())
```

Utils

GC-content:

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> from Bio.SeqUtils import GC
>>> my_seq = Seq('GATCGATGGGCCTATATAGGATCGAAAATCGC', IUPAC.unambiguous_dna)
>>> GC(my_seq)
46.875
```

Contatenation:

```
>>> from Bio.Alphabet import IUPAC
>>> dna_seq1 = Bio.Seq.Seq("ACGT", IUPAC.unambiguous_dna)
>>> dna_seq2 = Bio.Seq.Seq("ACCA", IUPAC.unambiguous_dna)
>>> dna_seq1 + dna_seq2
Seq('ACGTACCA', IUPACUnambiguousDNA())
```

WARNING: The alphabets must be compatible!

Transcription

```
>>> coding_dna
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG',
                                    IUPACUnambiguousDNA())
>>> messenger_rna = coding_dna.transcribe()
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG',
                                    IUPACUnambiguousRNA())
```

Complete transcription from template DNA:

```
>>> template_dna
Seq('CTATCGGGCACCCTTTCAGCGGCCCATTACAATGGCCAT', ...)
>>> template_dna.reverse_complement().transcribe()
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', ...)
```

Reverse transcription:

```
>>> messenger_rna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', .)
>>> messenger_rna.back_transcribe()
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', ...)
```

Translation

```
>>> from Bio.Seq import Seq
>>> from Bio.Alphabet import IUPAC
>>> mrna = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG", ...)
>>> mrna
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', ...)
>>> mrna.translate()
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Works also directly from DNA!