tinyfasta Documentation

Release 0.1.0

Tjelvar Olsson

November 01, 2016

Contents

CHAPTER 1

Content

1.1 TinyFasta

Python package for working with biological sequences from FASTA files.

- Documentation: http://tinyfasta.readthedocs.io
- GitHub: https://github.com/tjelvar-olsson/tinyfasta
- PyPI: https://pypi.python.org/pypi/tinyfasta
- Free software: MIT License

1.1.1 Features

- Easy to use: intuitive API for parsing, searching and writing FASTA files
- Lightweight: no dependencies outside Python's standard library
- Cross-platform: Linux, Mac and Windows are all supported
- Works with with Python 2.7, 3.3, 3.4 and 3.5

1.1.2 Quick Guide

To install the TinyFasta package:

```
sudo pip install tinyfasta
```

To parse a FASTA file:

```
>>> from tinyfasta import FastaParser
>>> for fasta_record in FastaParser("tests/data/dummy.fasta"):
...     if fasta_record.description.contains('seq1'):
...         print(fasta_record)
...
>seq1|contains 2x78 A's
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

To create a FASTA record:

```
>>> from tinyfasta import FastaRecord
>>> sequence = "C" * 100
>>> fasta_record = FastaRecord.create("My Sequence", sequence)
>>> print(fasta_record)
>My Sequence
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCC
```

1.2 Installation

The *tinyfasta* package can be installed using pip.

```
sudo pip install tinyfasta
```

Alternatively, you can clone the package from GitHub and install it.

```
git clone git@github.com:tjelvar-olsson/tinyfasta.git
cd tinyfasta
sudo python setup.py install
```

1.3 Parsing FASTA files

To parse a FASTA file we make use of the *tinyfasta.FastaParser* class.

```
>>> from tinyfasta import FastaParser
```

To create a *tinyfasta.FastaParser* instance we simply need the path to the FASTA file of interest.

```
>>> fasta_parser = FastaParser('tests/data/dummy.fasta')
>>> fasta_parser.fpath
'tests/data/dummy.fasta'
```

We can then iterate over all the *tinyfasta.FastaRecord* instances in the FASTA file.

```
>>> for fasta_record in fasta_parser:
...     print(fasta_record)
...
>seq1|contains 2x78 A's
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>seq2|starts with ATTA motif in first line
ATTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
```

1.4 Finding FASTA records

To find specific FASTA records one can simply iterate over the individual records in a particular FASTA file and check if the description and/or sequence contains a particular string or regular expression. Let us therefore start by creating a *tinyfasta.FastaParser* instance.

```
>>> from tinyfasta import FastaParser
>>> fasta_parser = FastaParser('tests/data/dummy.fasta')
```

1.4.1 Matching based on the description line

Now let us look for a FASTA record where the description contains the string `seq1`.

```
>>> for fasta_record in fasta_parser:
...     if fasta_record.description.contains('seq1'):
...         print(fasta_record)
...
>seq1|contains 2x78 A's
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Suppose we wanted to find all the FASTA records where the description line started with `>seq1|`, `>seq2|` or `>seq3|`. This query can be expressed using the regular expression below.

```
>>> import re
>>> search_term = re.compile(r'^>seq[1-3]\|')
```

We can use compiled regular expression to identify FASTA records of interest.

```
>>> for fasta_record in fasta_parser:
...     if fasta_record.description.contains(search_term):
...         print(fasta_record)
...
>seq1|contains 2x78 A's
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>seq2|starts with ATTA motif in first line
ATTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>seq3|ends with ATTA motif in second line
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATTA
```

1.4.2 Matching based on the sequence

We can use a similar approach to check if a *tinyfasta.FastaRecord* contains a sequence motif.

Let us first look for records containing a simple `ATTA` motif.

```
>>> for fasta_record in fasta_parser:
...     if fasta_record.sequence.contains('ATTA'):
...         print(fasta_record)
...
>seq2|starts with ATTA motif in first line
ATTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>seq3|ends with ATTA motif in second line
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATTA
>seq4|contains ATTA motif in middle of first line
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
>seq5|contains ATTA motif split over two lines
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAT
TAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

More complicated sequence motifs can be searched for by compiling regular expressions. Suppose we wanted to be able to identify any of the sequences below:

```
ACCCA
ACCTA
ACTTA
ATTTA
ATTCA
ATCCA
```

This could be achieved with the regular expression `A[C,T]{3}A`.

```
>>> motif = re.compile(r"A[C,T]{3}A")
```

Now let us find all the FASTA records that contain this motif.

```
>>> for fasta_record in fasta_parser:
...     if fasta_record.sequence.contains(motif):
...         print(fasta_record)
...
>seq7|contains ACCCA motif
AAAAAAAAAAAAAAAAAAAAAAAAAAAACCCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>seq8|contains ATTTA motif
AAAAAAAAAAAAAAAAAAAAAAAAAAAATTTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

1.4.3 Matching based on the sequence length

The `__len__()` magic method of both the *tinyfasta.Sequence* and *tinyfasta.FastaRecord* classes return the length of the biological sequence. One can therefore use Python's built-in `len()` function when looking for sequences of a particular length.

For example suppose we wanted to find all the sequences with fewer than 80 bases.

```
>>> for fasta_record in fasta_parser:
...     if len(fasta_record) < 80:
...         print(fasta_record)
...
>seq7|contains ACCCA motif
AAAAAAAAAAAAAAAAAAAAAAAAAAAACCCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>seq8|contains ATTTA motif
AAAAAAAAAAAAAAAAAAAAAAAAAAAATTTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

1.5 Creating FASTA records

There are two ways of creating *tinyfasta.FastaRecord* instances. We can create them from a description and a long sequence string or we can build them up from a description and several sequence strings. The latter approach is used internally by the *tinyfasta.FastaParser*.

1.5.1 Using a long sequence string

Let us import the *tinyfasta.FastaRecord* class and create a description and sequence strings.

```
>>> from tinyfasta import FastaRecord
>>> description = 'My Sequence'
>>> sequence = 'C' * 500
```

We can now create a *tinyfasta.FastaRecord* from the description and sequence strings by using the *tinyfasta.FastaRecord.create()* static method.

```
>>> from tinyfasta import FastaRecord
>>> fasta_record = FastaRecord.create(description, sequence)
```

Let us print out the record to verify what we got.

```
>>> print(fasta_record)
>My Sequence
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCC
```

1.5.2 Using several sequence strings

However, suppose that we wanted to create a *tinyfasta.FastaRecord* from a file containing the input sequence split over several lines. In this scenario we can simply add the sequence lines one by one.

Let us create a *tinyfasta.FastaRecord* to add the sequence lines to.

```
>>> fasta_record = FastaRecord('Yet Another Record')
```

Now we can start adding sequence lines to it.

```
>>> fasta_record.add_sequence_line("AAAAAAAA")
>>> fasta_record.add_sequence_line("TTTTTTTTTTTT")
>>> fasta_record.add_sequence_line("CCCCCC")
>>> fasta_record.add_sequence_line("GGGGGGGGGGGGGGG")
```

Note that by default the string representation of the *tinyfasta.FastaRecord* will contain the original sequence line splits.

```
>>> print(fasta_record)
>Yet Another Record
AAAAAAAA
TTTTTTTTTTTT
CCCCCC
GGGGGGGGGGGGGGG
```

However, using the `tinyfasta.FastaRecord.format_sequence_line_length()` function we can standardised line length.

```
>>> fasta_record.sequence.format_line_length(30)
>>> print(fasta_record)
>Yet Another Record
AAAAAAAATTTTTTTTTTTTCCCCCCGGGG
GGGGGGGGGGG
```

1.6 API

Package for parsing and generating FASTA files of biological sequences.

Use the *tinyfasta.FastaParser* class to parse FASTA files.

To generate FASTA files use the *tinyfasta.FastaRecord.create()* static method to create *tinyfasta.FastaRecord* instances, which can be written to file.

class tinyfasta.**Sequence**
Class representing a biological sequence.

add_sequence_line(*sequence_line*)
Add a sequence line to the *tinyfasta.Sequence* instance.

This function can be called more than once. Each time the function is called the *tinyfasta.Sequence* is extended by the sequence line provided.

Parameters **sequence_line** – string representing (part of) a sequence

format_line_length(*line_length=80*)
Format line length used to represent the sequence.

The full sequence is stored as list of shorter sequences. These shorter sequences are used verbatim when writing out the *tinyfasta.FastaRecord* over several lines.

Parameters **line_length** – length of the sequences used to make up the full sequence

class tinyfasta.**FastaRecord**(*description*)
Class representing a FASTA record.

class **Description**(*description*)
Description line in a *tinyfasta.FastaRecord*.

update(*description*)
Update the content of the description.

This function can be used to replace the existing description with a new one.
Parameters **description** – new description string

FastaRecord.**add_sequence_line**(*sequence_line*)
Add a sequence line to the *tinyfasta.FastaRecord* instance.

This function can be called more than once. Each time the function is called the tinyfasta.sequence is extended by the sequence line provided.

Parameters **sequence_line** – string representing (part of) a sequence

static FastaRecord.**create**(*description*, *sequence*)
Return a FastaRecord.

Parameters

- **description** – description string
- **sequence** – full sequence string

Returns *tinyfasta.FastaRecord*

class tinyfasta.**FastaParser**(*fpath*)
Class for parsing FASTA files.

t

A

C

F

S

T

U