bioutils

bioutils Contributors

May 05, 2021

CONTENTS:

bioutils provides common utilities and lookup tables for bioinformatics.

- bioutils.accessions – parse accessions, infer namespaces
- bioutils.assemblies – Human assembly information (from NCBI/GRCh)
- bioutils.cytobands – map cytobands to coordinates (from UCSC cytoband tables)
- bioutils.digests – implementations of various digests
- bioutils.normalize – allele normalization (left shuffle, right shuffle, expanded, vcf)

To use an E-Utilities API key run add it to an environment variable called *ncbi_api_key* and it will be used in the E-Utilities request.

CHAPTER

ONE

BIOUTILS PACKAGE

1.1 Submodules

1.1.1 bioutils.accessions module

Simple routines to deal with accessions, identifiers, etc.

Biocommons terminology: an identifier is composed of a *namespace* and an *accession*. The namespace is a string, composed of any character other than colon (:). The accession is a string without character set restriction. An accession is expected to be unique within the namespace; there is no expectation of uniqueness of accessions across namespaces.

```
Identifier := <Namespace, Accession>
```

```
Namespace := [^:]+
```

```
Accession := \w+
```

Some sample serializations of Identifiers:

```
json:  {"namespace":  "RefSeq", "accession":  "NM_000551.3"}
```

```
xml:  <Identifier namespace="RefSeq" accession="NM_000551.3"/>
```

```
string:  "RefSeq:NM_000551.3"
```

The string form may be used as a CURIE, in which case the document in which the CURIE is used must contain a map of `{namespace : uri}`.

bioutils.accessions.**chr22XY**(*c*)

Reformats chromosome to be of the form Chr1, . . . , Chr22, ChrX, ChrY, etc.

Parameters **c** (*str or int*) – A chromosome.

Returns The reformatted chromosome.

Return type str

Examples

```
>>> chr22XY('1')
'chr1'
```

```
>>> chr22XY(1)
'chr1'
```

```
>>> chr22XY('chr1')
'chr1'
```

```
>>> chr22XY(23)
'chrX'
```

```
>>> chr22XY(24)
'chrY'
```

```
>>> chr22XY("X")
'chrX'
```

```
>>> chr22XY("23")
'chrX'
```

```
>>> chr22XY("M")
'chrM'
```

bioutils.accessions.**coerce_namespace**(*ac*)

Prefixes accession with inferred namespace if not present.

Intended to be used to promote consistent and unambiguous accession identifiers.

Parameters **ac** (*str*) – The accession, with or without namespace prefixed.

Returns An identifier of the form "{namespace}:{acession}"

Return type str

Raises **ValueError** – if accession sytax does not match the syntax of any namespace.

Examples

```
>>> coerce_namespace("refseq:NM_01234.5")
'refseq:NM_01234.5'
```

```
>>> coerce_namespace("NM_01234.5")
'refseq:NM_01234.5'
```

```
>>> coerce_namespace("bogus:QQ_01234.5")
'bogus:QQ_01234.5'
```

```
>>> coerce_namespace("QQ_01234.5")
Traceback (most recent call last):
...
ValueError: Could not infer namespace for QQ_01234.5
```

bioutils.accessions.**infer_namespace**(*ac*)

Infers a unique namespace from an accession, if one exists.

Parameters **ac** (*str*) – An accession, without the namespace prefix.

Returns

The unique namespace corresponding to accession syntax, if only one is inferred. None if the accesssion sytax does not match any namespace.

Return type str or None

Raises ***BioutilsError*** – If multiple namespaces match the syntax of the accession.

Examples

```
>>> infer_namespace("ENST00000530893.6")
'ensembl'
```

```
>>> infer_namespace("NM_01234.5")
'refseq'
```

```
>>> infer_namespace("A2BC19")
'uniprot'
```

Disbled because Python 2 and 3 handles exceptions differently.

```
>>> infer_namespace("P12345")
Traceback (most recent call last):
...
bioutils.exceptions.BioutilsError: Multiple namespaces possible for P12345
```

```
>>> infer_namespace("BOGUS99") is None
True
```

bioutils.accessions.**infer_namespaces**(*ac*)

Infers namespaces possible for a given accession, based on syntax.

Parameters **ac** (*str*) – An accession, without the namespace prefix.

Returns A list of namespaces matching the accession, possibly empty.

Return type list of str

Examples

```
>>> infer_namespaces("ENST00000530893.6")
['ensembl']
```

```
>>> infer_namespaces("ENST00000530893")
['ensembl']
```

```
>>> infer_namespaces("ENSQ00000530893")
[]
```

```
>>> infer_namespaces("NM_01234")
['refseq']
```

```
>>> infer_namespaces("NM_01234.5")
['refseq']
```

```
>>> infer_namespaces("NQ_01234.5")
[]
```

```
>>> infer_namespaces("A2BC19")
['uniprot']
```

```
>>> sorted(infer_namespaces("P12345"))
['insdc', 'uniprot']
```

```
>>> infer_namespaces("A0A022YWF9")
['uniprot']
```

bioutils.accessions.**prepend_chr**(*chr*)

Prepends chromosome with 'chr' if not present.

Users are strongly discouraged from using this function. Adding a 'chr' prefix results in a name that is not consistent with authoritative assembly records.

Parameters **chr** (*str*) – The chromosome.

Returns The chromosome with 'chr' prepended.

Return type str

Examples

```
>>> prepend_chr('22')
'chr22'
```

```
>>> prepend_chr('chr22')
'chr22'
```

bioutils.accessions.**strip_chr**(*chr*)

Removes the 'chr' prefix if present.

Parameters **chr** (*str*) – The chromosome.

Returns The chromosome without a 'chr' prefix.

Return type str

Examples

```
>>> strip_chr('22')
'22'
```

```
>>> strip_chr('chr22')
'22'
```

1.1.2 bioutils.assemblies module

Creates dictionaries of genome assembly data as provided by

ftp://ftp.ncbi.nlm.nih.gov/genomes/ASSEMBLY_REPORTS/All/*.assembly.txt

Assemblies are stored in json files with the package in `_data/assemblies/`. Those files are built with sbin/assembly-to-json, also in this package.

Definitions:

- accession `ac`: symbol used to refer to a sequence (e.g., NC_000001.10)
- name: human-label (e.g., '1', 'MT', 'HSCHR6_MHC_APD_CTG1') that refers to a sequence, unique within some domain (e.g., GRCh37.p10)
- chromosome (chr): subset of names that refer to chromosomes 1..22, X, Y, MT
- aliases: list of other names; uniqueness unknown

Note: Some users prefer using a 'chr' prefix for chromosomes and some don't. Some prefer upper case and others prefer lower. This rift is unfortunate and creates unnecessary friction in sharing data. You say TO-my-to and I say TO-mah-to doesn't apply here. This code favors using the authoritative names exactly as defined in the assembly records. Users are encouraged to use sequence names verbatim, without prefixes or case changes.

`bioutils.assemblies.`**`get_assemblies`**(*names=[]*)

Retrieves data from multiple assemblies.

If assemblies are not specified, retrieves data from all available ones.

Parameters **`names`** (*`list of str, optional`*) – The names of the assemblies to retrieve data for.

Returns

A dictionary of the form `{assembly_name, : assembly_data}`, where the values are the dictionaries of assembly data as described in `get_assembly()`.

Return type dict

Examples

```
>>> assemblies = get_assemblies(names=['GRCh37.p13'])
>>> assy = assemblies['GRCh37.p13']
```

```
>>> assemblies = get_assemblies()
>>> 'GRCh38.p2' in assemblies
True
```

bioutils.assemblies.**get_assembly**(*name*)

Retreives the assembly data for a given assembly.

Parameters **name** (*str*) – The name of the assembly to retrieve data for.

Returns A dictionary of the assembly data. See examples for details.

Return type dict

Examples

```
>>> assy = get_assembly('GRCh37.p13')
```

```
>>> assy['name']
'GRCh37.p13'
```

```
>>> assy['description']
'Genome Reference Consortium Human Build 37 patch release 13 (GRCh37.p13)'
```

```
>>> assy['refseq_ac']
'GCF_000001405.25'
```

```
>>> assy['genbank_ac']
'GCA_000001405.14'
```

```
>>> len(assy['sequences'])
297
```

```
>>> import pprint
>>> pprint.pprint(assy['sequences'][0])
{'aliases': ['chr1'],
'assembly_unit': 'Primary Assembly',
'genbank_ac': 'CM000663.1',
'length': 249250621,
'name': '1',
'refseq_ac': 'NC_000001.10',
'relationship': '=',
'sequence_role': 'assembled-molecule'}
```

bioutils.assemblies.**get_assembly_names**()

Retrieves available assemblies from the _data/assemblies directory.

Returns The names of the available assemblies.

Return type list of str

Examples

```
>>> assy_names = get_assembly_names()
```

```
>>> 'GRCh37.p13' in assy_names
True
```

bioutils.assemblies.**make_ac_name_map**(*assy_name*, *primary_only=False*)

Creates a map from accessions to sequence names for a given assembly.

Parameters

- **assy_name** (*str*) – The name of the assembly to make a map for.
- **primary_only** (*bool, optional*) – Whether to include only primary sequences. Defaults to False.

Returns

A dictionary of the form {accesssion : sequence_name} for accessions in the given assembly, where accession and sequence_name are strings.

Return type dict

Examples

```
>>> grch38p5_ac_name_map = make_ac_name_map('GRCh38.p5')
>>> grch38p5_ac_name_map['NC_000001.11']
'1'
```

bioutils.assemblies.**make_name_ac_map**(*assy_name*, *primary_only=False*)

Creates a map from sequence names to accessions for a given assembly.

Parameters

- **assy_name** (*str*) – The name of the assembly to make a map for.
- **primary_only** (*bool, optional*) – Whether to include only primary sequences. Defaults to False.

Returns

A dictionary of the form {sequence_name : accession} for sequences in the given assembly, Where sequence_name and accession are both strings.

Return type dict

Examples

```
>>> grch38p5_name_ac_map = make_name_ac_map('GRCh38.p5')
>>> grch38p5_name_ac_map['1']
'NC_000001.11'
```

1.1.3 bioutils.coordinates module

Provides utilities for interconverting between coordinate systems especially as used by the hgvs code. The three systems are:

```
                : A : C : G : T : A : C :
human/hgvs  h   :-3 :-2 :-1 : 1 : 2 : 3 :
continuous  c   :-2 :-1 : 0 : 1 : 2 : 3 :
interbase   i  -3  -2  -1   0   1   2   3
```

Human/hgvs coordinates are the native coordinates used by the HGVS recommendations. The coordinates are 1-based, inclusive, and refer to the nucleotides; there is no 0.

Continuous coordinates are similar to hgvs coordinates, but adds 1 to all negative values so that there is no discontinuity between -1 and 1 (as there is with HGVS).

Interbase coordinates refer to the zero-width junctions between nucleotides. The main advantage of interbase coordinates is that there are no corner cases in the specification of intervals used for insertions and deletions as there is with numbering systems that refer to nucleotides themselves. Numerically, interbase intervals are 0-based, left-closed, and right-open. Beacuse referring to a single interbase coordinate is not particularly meaningful, interbase coordinates are always passed as start,end pairs.

Because it's easy to confuse these coordinates in code, `_h`, `_c`, and `_i` suffixes are often used to clarify variables.

For code clarity, this module provides functions that interconvert *intervals* specified in each of the coordinate systems.

bioutils.coordinates.**strand_int_to_pm**(*i*)

Converts 1 and -1 to '+' and '-' respectively.

Parameters **i** (*int*) –

Returns '+' if i == 1, '-' if i == -1, otherwise None.

Return type str

Examples

```
>>> strand_int_to_pm(1)
'+'
>>> strand_int_to_pm(-1)
'-'
>>> strand_int_to_pm(42)
```

bioutils.coordinates.**strand_pm**(*i*)

Converts 1 and -1 to '+' and '-' respectively.

Parameters **i** (*int*) –

Returns '+' if i == 1, '-' if i == -1, otherwise None.

Return type str

Examples

```
>>> strand_int_to_pm(1)
'+'
>>> strand_int_to_pm(-1)
'-'
>>> strand_int_to_pm(42)
```

bioutils.coordinates.**strand_pm_to_int**(*s*)

Converts '+' and '-' to 1 and -1, respectively.

Parameters **s** (*string*) –

Returns 1 if s == '+', -1 if s == '-', otherwise None.

Return type int

Examples

```
>>> strand_pm_to_int('+')
1
>>> strand_pm_to_int('-')
-1
>>> strand_pm_to_int('arglefargle')
```

1.1.4 bioutils.cytobands module

```
./sbin/ucsc-cytoband-to-json cytoband-hg38.txt.gz | gzip -c >bioutils/_data/
cytobands/ucsc-hg38.json.gz
```

bioutils.cytobands.**get_cytoband_map**(*name*)

Retrives a cytoband by name.

Parameters **name** (*str*) – The name of the cytoband to retrieve.

Returns A dictionary of the cytoband data.

Return type dict

Examples

```
>>> map = get_cytoband_map("ucsc-hg38")
>>> map["1"]["p32.2"]
[55600000, 58500000, 'gpos50']
```

bioutils.cytobands.**get_cytoband_maps**(*names=[]*)

Retrieves data from multiple cytobands.

If cytobands are not specified, retrieves data from all available ones.

Parameters **names** (*list of str, optional*) – The names of cytobands to retrieve data for.

Returns A dictionary of the form `{cytoband_name, cytoband_data}`.

Return type dict

Examples

```
>>> maps = get_cytoband_maps()
>>> maps["ucsc-hg38"]["1"]["p32.2"]
[55600000, 58500000, 'gpos50']
>>> maps["ucsc-hg19"]["1"]["p32.2"]
[56100000, 59000000, 'gpos50']
```

bioutils.cytobands.**get_cytoband_names**()

Retrieves available cytobands from the _data/cytobands directory.

Returns The names of the available cytobands.

Return type list of str

Examples

```
>>> sorted(get_cytoband_names())
['ucsc-hg19', 'ucsc-hg38']
```

1.1.5 bioutils.digest module

class bioutils.digest.**Digest**

Bases: bytes

Represents a sliceable binary digest, with support for encoding and decoding using printable characters.

Supported encoding and decodings are::

- base64
- base64url
- hex (aka base16)

The Base64 specification (https://tools.ietf.org/html/rfc4648#page-7) defines base64 and a URL-safe variant called base64url.

"Stringified" Digest objects use URL-safe base64 encodings.

```
>>> import hashlib
```

```
>>> b = hashlib.sha512().digest()
>>> len(b)
64
```

```
>>> d = Digest(b)           # creation
>>> str(d)                  # returns base64url
'z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXcg_SpIdNs6c5H0NE8XYXysP-DGNKHfuwvY7kxvUdBeoGlODJ6-
↪SfaPg=='
```

```
>>> d24 = d[:24]            # slice binary digest at first 24 bytes
>>> str(d24)
'z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXc'
```

encoding

```
>>> d.as_base64url()
'z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXcg_SpIdNs6c5H0NE8XYXysP-DGNKHfuwvY7kxvUdBeoGlODJ6-
↪SfaPg=='
>>> d.as_hex()

↪'cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877ee
↪'
```

decoding

```
>>> d == Digest.from_base64(d.as_base64())
True
>>> d == Digest.from_base64url(d.as_base64url())
True
>>> d == Digest.from_hex(d.as_hex())
True
```

as_base64()
Returns Digest as a base64-encoded string.

Returns base64 encoding of Digest.

Return type str

as_base64url()
Returns Digest as URL-safe, base64-encoded string.

Returns URL-safe base64 encoding of Digest.

Return type str

as_base64us()
Returns Digest as URL-safe, base64-encoded string.

Returns URL-safe base64 encoding of Digest.

Return type str

as_hex()
Returns Digest as hex string.

Returns A hex-encoding of Digest.

Return type str

static from_base64(*s*)
Returns Digest object initialized from a base64-encoded string.

Parameters **s** (*str*) – A base64-encoded digest string.

Returns A Digest object initialized from s.

Return type *Digest*

static from_base64url(*s*)
Returns Digest object initialized from a base64url string.

Parameters **s** (*str*) – A base64url-encoded digest string.

Returns A Digest object initialized from s.

Return type *Digest*

static from_base64us(*s*)
Returns Digest object initialized from a base64url string.

Parameters **s** (*str*) – A base64url-encoded digest string.

Returns A Digest object initialized from s.

Return type *Digest*

static from_hex(*s*)
returns Digest object initialized from hex string.

Parameters **s** (*str*) – A hex-encoded digest string.

Returns A Digest object initialized from s.

Return type *Digest*

1.1.6 bioutils.digests module

bioutils.digests.**seq_md5**(*seq*, *normalize=True*)
Converts sequence to unicode md5 hex digest.

Parameters

- **seq** (*str*) – A sequence.
- **normalize** (*bool, optional*) – Whether to normalize the sequence before conversion, i.e. to ensure representation as uppercase letters without whitespace or asterisks. Defaults to `True`.

Returns Unicode md5 hex digest representation of sequence.

Return type str

Examples

```
>>> seq_md5('')
'd41d8cd98f00b204e9800998ecf8427e'
```

```
>>> seq_md5('ACGT')
'f1f8f4bf413b16ad135722aa4591043e'
```

```
>>> seq_md5('ACGT*')
'f1f8f4bf413b16ad135722aa4591043e'
```

```
>>> seq_md5(' A C G T ')
'f1f8f4bf413b16ad135722aa4591043e'
```

```
>>> seq_md5('acgt')
'f1f8f4bf413b16ad135722aa4591043e'
```

```
>>> seq_md5('acgt', normalize=False)
'db516c3913e179338b162b2476d1c23f'
```

bioutils.digests.**seq_seguid**(*seq*, *normalize=True*)
Converts sequence to seguid.

This seguid is compatible with BioPython's seguid.

Parameters

- **seq** (*str*) – A sequence.
- **normalize** (*bool, optional*) – Whether to normalize the sequence before conversion, i.e. to ensure representation as uppercase letters without whitespace or asterisks. Defaults to `True`.

Returns seguid representation of sequence.

Return type str

Examples

```
>>> seq_seguid('')
'2jmj7l5rSw0yVb/vlWAYkK/YBwk'
```

```
>>> seq_seguid('ACGT')
'IQiZThf2zKn/I1KtqStlEdsHYDQ'
```

```
>>> seq_seguid('acgt')
'IQiZThf2zKn/I1KtqStlEdsHYDQ'
```

```
>>> seq_seguid('acgt', normalize=False)
'lII0AoG1/I8qKY271rgv5CFZtsU'
```

bioutils.digests.**seq_seqhash**(*seq*, *normalize=True*)

Converts sequence to 24-byte Truncated Digest.

Parameters

- **seq** (*str*) – A sequence.
- **normalize** (*bool, optional*) – Whether to normalize the sequence before conversion, i.e. to ensure representation as uppercase letters without whitespace or asterisks. Defaults to `True`.

Returns 24-byte Truncated Digest representation of sequence.

Return type str

Examples

```
>>> seq_seqhash("")
'z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXc'
```

```
>>> seq_seqhash("ACGT")
'aKF498dAxcJAqme6QYQ7EZ07-fiw8Kw2'
```

```
>>> seq_seqhash("acgt")
'aKF498dAxcJAqme6QYQ7EZ07-fiw8Kw2'
```

```
>>> seq_seqhash("acgt", normalize=False)
'eFwawHHdibaZBDcs9kW3gm31h1NNJcQe'
```

bioutils.digests.**seq_sha1**(*seq*, *normalize=True*)

Converts sequence to unicode sha1 hexdigest.

Parameters

- **seq** (*str*) – A sequence.
- **normalize** (*bool, optional*) – Whether to normalize the sequence before conversion, i.e. to ensure representation as uppercase letters without whitespace or asterisks before encoding. Defaults to `True`.

Returns Unicode sha1 hexdigest representation of sequence.

Return type str

Examples

```
>>> seq_sha1('')
'da39a3ee5e6b4b0d3255bfef95601890afd80709'
```

```
>>> seq_sha1('ACGT')
'2108994e17f6cca9ff2352ada92b6511db076034'
```

```
>>> seq_sha1('acgt')
'2108994e17f6cca9ff2352ada92b6511db076034'
```

```
>>> seq_sha1('acgt', normalize=False)
'9482340281b5fc8f2a298dbbd6b82fe42159b6c5'
```

bioutils.digests.**seq_sha512**(*seq*, *normalize=True*)

Converts sequence to unicode sha512 hexdigest.

Parameters

- **seq** (*str*) – A sequence.
- **normalize** (*bool, optional*) – Whether to normalize the sequence before conversion, i.e. to ensure representation as uppercase letters without whitespace or asterisks. Defaults to `True`.

Returns Unicode sha512 hexdigest representation of sequence.

Return type str

Examples

```
>>> seq_sha512('')

↪'cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877ee
↪'
```

```
>>> seq_sha512('ACGT')

↪'68a178f7c740c5c240aa67ba41843b119d3bf9f8b0f0ac36cf701d26672964efbd536d197f51ce634fc70634d1eef
↪'
```

```
>>> seq_sha512('acgt')

↪'68a178f7c740c5c240aa67ba41843b119d3bf9f8b0f0ac36cf701d26672964efbd536d197f51ce634fc70634d1eef
↪'
```

(continues on next page)

(continued from previous page)

```
>>> seq_sha512('acgt', normalize=False)

↪'785c1ac071dd89b69904372cf645b7826df587534d25c41edb2862e54fb2940d697218f2883d2bf1a11cdaee658c7
↪'
```

bioutils.digests.**seq_vmc_id**(*seq*, *normalize=True*)

Converts sequence to VMC id.

See https://github.com/ga4gh/vmc

Parameters

- **seq** (*str*) – A sequence.
- **normalize** (*bool, optional*) – Whether to normalize the sequence before conversion, i.e. to ensure representation as uppercase letters without whitespace or asterisks. Defaults to `True`.

Returns VMC id representation of sequence.

Return type str

Examples

```
>>> seq_vmc_id("")
'VMC:GS_z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXc'
```

```
>>> seq_vmc_id("ACGT")
'VMC:GS_aKF498dAxcJAqme6QYQ7EZ07-fiw8Kw2'
```

```
>>> seq_vmc_id("acgt")
'VMC:GS_aKF498dAxcJAqme6QYQ7EZ07-fiw8Kw2'
```

```
>>> seq_vmc_id("acgt", normalize=False)
'VMC:GS_eFwawHHdibaZBDcs9kW3gm31h1NNJcQe'
```

bioutils.digests.**seq_vmc_identifier**(*seq*, *normalize=True*)

Converts sequence to VMC identifier (record).

See https://github.com/ga4gh/vmc

Parameters

- **seq** (*str*) – A sequence.
- **normalize** (*bool, optional*) – Whether to normalize the sequence before conversion, i.e. to ensure representation as uppercase letters without whitespace or asterisks. Defaults to `True`.

Returns VMC identifier (record) representation of sequnce.

Return type str

Examples

```
>>> seq_vmc_identifier("") == {'namespace': 'VMC', 'accession': 'GS_
↪z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXc'}
True
```

```
>>> seq_vmc_identifier("ACGT") == {'namespace': 'VMC', 'accession': 'GS_
↪aKF498dAxcJAqme6QYQ7EZ07-fiw8Kw2'}
True
```

```
>>> seq_vmc_identifier("acgt") == {'namespace': 'VMC', 'accession': 'GS_
↪aKF498dAxcJAqme6QYQ7EZ07-fiw8Kw2'}
True
```

```
>>> seq_vmc_identifier("acgt", normalize=False) == {'namespace': 'VMC', 'accession
↪': 'GS_eFwawHHdibaZBDcs9kW3gm31h1NNJcQe'}
True
```

1.1.7 bioutils.exceptions module

exception bioutils.exceptions.**BioutilsError**
Bases: Exception

Root exception for all bioutils exceptions

1.1.8 bioutils.normalize module

Provides functionality for normalizing alleles, ensuring comparable representations.

class bioutils.normalize.**NormalizationMode**(*value*)
Bases: enum.Enum

Enum passed to normalize to select the normalization mode.

EXPAND
Normalize alleles to maximal extent both left and right.

LEFTSHUFFLE
Normalize alleles to maximal extent left.

RIGHTSHUFFLE
Normalize alleles to maximal extent right.

TRIMONLY
Only trim the common prefix and suffix of alleles.

VCF
Normalize with VCF.

EXPAND = 1

LEFTSHUFFLE = 2

RIGHTSHUFFLE = 3

TRIMONLY = 4

VCF = 5

bioutils.normalize.**normalize**(*sequence*, *interval*, *alleles*, *mode=<NormalizationMode.EXPAND: 1>*, *bounds=None*, *anchor_length=0*)

Normalizes the alleles that co-occur on sequence at interval, ensuring comparable representations.

Parameters

- **sequence** (*str or iterable*) – The reference sequence; must support indexing and __getitem__.
- **interval** (*2-tuple of int*) – The location of alleles in the reference sequence as `(start, end)`. Interbase coordinates.
- **alleles** (*iterable of str*) – The sequences to be normalized. The first element corresponds to the reference sequence being unchanged and must be None.
- **bounds** (*2-tuple of int, optional*) – Maximal extent of normalization left and right. Must be provided if sequence doesn't support __len__. Defaults to `(0, len(sequence))`.
- **mode** (*NormalizationMode Enum or string, optional*) – A NormalizationMode Enum or the corresponding string. Defaults to `EXPAND`.
- **anchor** (*int, optional*) – number of flanking residues left and right. Defaults to `0`.

Returns `(new_interval, [new_alleles])`

Return type tuple

Raises

- **ValueError** – If normalization mode is VCF and *anchor_length* is nonzero.
- **ValueError** – If the interval start is greater than the end.
- **ValueError** – If the first (reference) allele is not *None*.
- **ValueError** – If there are not at least two distinct alleles.

Examples

```
>>> sequence = "CCCCCCCCACACACACACTAGCAGCAGCA"
>>> normalize(sequence, interval=(22,25), alleles=(None, "GC", "AGCAC"), mode=
↪'TRIMONLY')
((22, 24), ('AG', 'G', 'AGCA'))
```

```
>>> normalize(sequence, interval=(22, 22), alleles=(None, 'AGC'), mode=
↪'RIGHTSHUFFLE')
((29, 29), ('', 'GCA'))
```

```
>>> normalize(sequence, interval=(22, 22), alleles=(None, 'AGC'), mode='EXPAND')
((19, 29), ('AGCAGCAGCA', 'AGCAGCAGCAGCA'))
```

bioutils.normalize.**roll_left**(*sequence*, *alleles*, *ref_pos*, *bound*)

Determines common distance all alleles can be rolled (circularly permuted) left within the reference sequence without altering it.

Parameters

- **sequence** (*str*) – The reference sequence.
- **alleles** (*list of str*) – The sequences to be normalized.

- **ref_pos** (*int*) – The beginning index for rolling.
- **bound** (*int*) – The lower bound index in the reference sequence for normalization, hence also for rolling.

Returns The distance that the alleles can be rolled.

Return type int

bioutils.normalize.**roll_right**(*sequence*, *alleles*, *ref_pos*, *bound*)

Determines common distance all alleles can be rolled (circularly permuted) right within the reference sequence without altering it.

Parameters

- **sequence** (*str*) – The reference sequence.
- **alleles** (*list of str*) – The sequences to be normalized.
- **ref_pos** (*int*) – The start index for rolling.
- **bound** (*int*) – The upper bound index in the reference sequence for normalization, hence also for rolling.

Returns The distance that the alleles can be rolled

Return type int

bioutils.normalize.**trim_left**(*alleles*)

Removes common prefix of given alleles.

Parameters **alleles** (*list of str*) – A list of alleles.

Returns (number_trimmed, [new_alleles]).

Return type tuple

Examples

```
>>> trim_left(["","AA"])
(0, ['', 'AA'])
```

```
>>> trim_left(["A","AA"])
(1, ['', 'A'])
```

```
>>> trim_left(["AT","AA"])
(1, ['T', 'A'])
```

```
>>> trim_left(["AA","AA"])
(2, ['', ''])
```

```
>>> trim_left(["CAG","CG"])
(1, ['AG', 'G'])
```

bioutils.normalize.**trim_right**(*alleles*)

Removes common suffix of given alleles.

Parameters **alleles** (*list of str*) – A list of alleles.

Returns (number_trimmed, [new_alleles]).

Return type tuple

Examples

```
>>> trim_right(["","AA"])
(0, ['', 'AA'])
```

```
>>> trim_right(["A","AA"])
(1, ['', 'A'])
```

```
>>> trim_right(["AT","AA"])
(0, ['AT', 'AA'])
```

```
>>> trim_right(["AA","AA"])
(2, ['', ''])
```

```
>>> trim_right(["CAG","CG"])
(1, ['CA', 'C'])
```

1.1.9 bioutils.seqfetcher module

Provides sequence fetching from NCBI and Ensembl.

bioutils.seqfetcher.**fetch_seq**(*ac*, *start_i=None*, *end_i=None*)

Fetches sequences and subsequences from NCBI eutils and Ensembl REST interfaces.

Parameters

- **ac** (*str*) – The accession of the sequence to fetch.
- **start_i** (*int, optional*) – The start index (interbase coordinates) of the subsequence to fetch. Defaults to None. It is recommended to retrieve a subsequence by providing an index here, rather than by Python slicing the whole sequence.
- **end_i** (*int, optional*) – The end index (interbase coordinates) of the subsequence to fetch. Defaults to None. It is recommended to retrieve a subsequence by providing an index here, rather than by Python slicing the whole sequence.

Returns The requested sequence.

Return type str

Raises

- **RuntimeError** – If the syntax doesn't match that of any of the databases.
- **RuntimeError** – If the request to the database fails.

Examples

```
>>> len(fetch_seq('NP_056374.2'))
1596
```

```
>>> fetch_seq('NP_056374.2',0,10)    # This!
'MESRETLSSS'
```

```
>>> fetch_seq('NP_056374.2')[0:10]  # Not this!
'MESRETLSSS'
```

Providing intervals is especially important for large sequences:

```
>>> fetch_seq('NC_000001.10',2000000,2000030)
'ATCACACGTGCAGGAACCCTTTTCCAAAGG'
```

This call will pull back 30 bases plus overhead; without the # interval, one would receive 250MB of chr1 plus overhead!

Essentially any RefSeq, Genbank, BIC, or Ensembl sequence may be # fetched.

```
>>> fetch_seq('NM_9.9')
Traceback (most recent call last):
...
RuntimeError: No sequence available for NM_9.9
```

```
>>> fetch_seq('QQ01234')
Traceback (most recent call last):
...
RuntimeError: No sequence fetcher for QQ01234
```

1.1.10 bioutils.sequences module

Simple functions and lookup tables for nucleic acid and amino acid sequences.

bioutils.sequences.**aa1_to_aa3**(*seq*)

Converts string of 1-letter amino acids to 3-letter amino acids.

Should only be used if the format of the sequence is known; otherwise use `aa_to_aa3()`.

Parameters **seq** (*str*) – An amino acid sequence as 1-letter amino acids.

Returns The sequence as 3-letter amino acids.

Return type str

Raises **KeyError** – If the sequence is not of 1-letter amino acids.

Examples

```
>>> aa1_to_aa3("CATSARELAME")
'CysAlaThrSerAlaArgGluLeuAlaMetGlu'
```

```
>>> aa1_to_aa3(None)
```

bioutils.sequences.**aa3_to_aa1**(*seq*)

Converts string of 3-letter amino acids to 1-letter amino acids.

Should only be used if the format of the sequence is known; otherwise use `aa_to_aa1()`.

Parameters **seq** (*str*) – An amino acid sequence as 3-letter amino acids.

Returns The sequence as 1-letter amino acids.

Return type str

Raises **KeyError** – If the sequence is not of 3-letter amino acids.

Examples

```
>>> aa3_to_aa1("CysAlaThrSerAlaArgGluLeuAlaMetGlu")
'CATSARELAME'
```

```
>>> aa3_to_aa1(None)
```

bioutils.sequences.**aa_to_aa1**(*seq*)

Coerces string of 1- or 3-letter amino acids to 1-letter representation.

Parameters **seq** (*str*) – An amino acid sequence.

Returns The sequence as one of 1-letter amino acids.

Return type str

Examples

```
>>> aa_to_aa1("CATSARELAME")
'CATSARELAME'
```

```
>>> aa_to_aa1("CysAlaThrSerAlaArgGluLeuAlaMetGlu")
'CATSARELAME'
```

```
>>> aa_to_aa1(None)
```

bioutils.sequences.**aa_to_aa3**(*seq*)

Coerces string of 1- or 3-letter amino acids to 3-letter representation.

Parameters **seq** (*str*) – An amino acid sequence.

Returns The sequence as one of 3-letter amino acids.

Return type str

Examples

```
>>> aa_to_aa3("CATSARELAME")
'CysAlaThrSerAlaArgGluLeuAlaMetGlu'
```

```
>>> aa_to_aa3("CysAlaThrSerAlaArgGluLeuAlaMetGlu")
'CysAlaThrSerAlaArgGluLeuAlaMetGlu'
```

```
>>> aa_to_aa3(None)
```

bioutils.sequences.**complement**(*seq*)

Retrieves the complement of a sequence.

Parameters **seq** (*str*) – A nucleotide sequence.

Returns The complement of the sequence.

Return type str

Examples

```
>>> complement("ATCG")
'TAGC'
```

```
>>> complement(None)
```

bioutils.sequences.**elide_sequence**(*s*, *flank=5*, *elision='...'*)

Trims the middle of the sequence, leaving the right and left flanks.

Parameters

- **s** (*str*) – A sequence.
- **flank** (*int, optional*) – The length of each flank. Defaults to five.
- **elision** (*str, optional*) – The symbol used to represent the part trimmed. Defaults to '...'.
- **Returns** – str: The sequence with the middle replaced by `elision`.

Examples

```
>>> elide_sequence("ABCDEFGHIJKLMNOPQRSTUVWXYZ")
'ABCDE...VWXYZ'
```

```
>>> elide_sequence("ABCDEFGHIJKLMNOPQRSTUVWXYZ", flank=3)
'ABC...XYZ'
```

```
>>> elide_sequence("ABCDEFGHIJKLMNOPQRSTUVWXYZ", elision="..")
'ABCDE..VWXYZ'
```

```
>>> elide_sequence("ABCDEFGHIJKLMNOPQRSTUVWXYZ", flank=12)
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
>>> elide_sequence("ABCDEFGHIJKLMNOPQRSTUVWXYZ", flank=12, elision=".")
'ABCDEFGHIJKL.OPQRSTUVWXYZ'
```

bioutils.sequences.**looks_like_aa3_p**(*seq*)

Indicates whether a string looks like a 3-letter AA string.

Parameters **seq** (*str*) – A sequence.

Returns Whether the string is of the format of a 3-letter AA string.

Return type bool

bioutils.sequences.**normalize_sequence**(*seq*)

Converts sequence to normalized representation for hashing.

Essentially, removes whitespace and asterisks, and uppercases the string.

Parameters **seq** (*str*) – The sequence to be normalized.

Returns The sequence as a string of uppercase letters.

Return type str

Raises **RuntimeError** – If the sequence contains non-alphabetic characters (besides '*').

Examples

```
>>> normalize_sequence("ACGT")
'ACGT'
```

```
>>> normalize_sequence("  A C G T * ")
'ACGT'
```

```
>>> normalize_sequence("ACGT1")
Traceback (most recent call last):
...
RuntimeError: Normalized sequence contains non-alphabetic characters
```

bioutils.sequences.**replace_t_to_u**(*seq*)

Replaces the T's in a sequence with U's.

Parameters **seq** (*str*) – A nucleotide sequence.

Returns The sequence with the T's replaced by U's.

Return type str

Examples

```
>>> replace_t_to_u("ACGT")
'ACGU'
```

```
>>> replace_t_to_u(None)
```

bioutils.sequences.**replace_u_to_t**(*seq*)

Replaces the U's in a sequence with T's.

Parameters **seq** (*str*) – A nucleotide sequence.

Returns The sequence with the U's replaced by T's.

Return type str

Examples

```
>>> replace_u_to_t("ACGU")
'ACGT'
```

```
>>> replace_u_to_t(None)
```

bioutils.sequences.**reverse_complement**(*seq*)

Converts a sequence to its reverse complement.

Parameters **seq** (*str*) – A nucleotide sequence.

Returns The reverse complement of the sequence.

Return type str

Examples

```
>>> reverse_complement("ATCG")
'CGAT'
```

```
>>> reverse_complement(None)
```

bioutils.sequences.**translate_cds**(*seq*, *full_codons=True*, *ter_symbol='*'*)

Translates a DNA or RNA sequence into a single-letter amino acid sequence.

Uses the NCBI standard translation table.

Parameters

- **seq** (*str*) – A nucleotide sequence.
- **full_codons** (*bool, optional*) – If `True`, forces sequence to have length that is a multiple of 3 and raises an error otherwise. If False, `ter_symbol` will be added as the last amino acid. This corresponds to biopython's behavior of padding the last codon with N``s. Defaults to ``True.
- **ter_symbol** (*str, optional*) – Placeholder for the last amino acid if sequence length is not divisible by three and `full_codons` is False. Defaults to `'*'`

Returns The corresponding single letter amino acid sequence.

Return type str

Raises

- **ValueError** – If `full_codons` and the sequence is not a multiple of three.
- **ValueError** – If a codon is undefined in the table.

Examples

```
>>> translate_cds("ATGCGA")
'MR'
```

```
>>> translate_cds("AUGCGA")
'MR'
```

```
>>> translate_cds(None)
```

```
>>> translate_cds("")
''
```

```
>>> translate_cds("AUGCG")
Traceback (most recent call last):
...
ValueError: Sequence length must be a multiple of three
```

```
>>> translate_cds("AUGCG", full_codons=False)
'M*'
```

```
>>> translate_cds("ATGTAN")
'MX'
```

```
>>> translate_cds("CCN")
'P'
```

```
>>> translate_cds("TRA")
'*'
```

```
>>> translate_cds("TTNTA", full_codons=False)
'X*'
```

```
>>> translate_cds("CTB")
'L'
```

```
>>> translate_cds("AGM")
'X'
```

```
>>> translate_cds("GAS")
'X'
```

```
>>> translate_cds("CUN")
'L'
```

```
>>> translate_cds("AUGCGQ")
Traceback (most recent call last):
...
ValueError: Codon CGQ at position 4..6 is undefined in codon table
```

1.1.11 bioutils.vmc_digest module

bioutils.vmc_digest.**vmc_digest**(*data*, *digest_size=24*)

Returns the VMC Digest as a Digest object, which has both bytes and string (URL-safe, Base 64) representations.

```
>>> d = vmc_digest("")
```

```
>>> str(d)
'z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXc'
```

```
>>> len(d), len(str(d))
(24, 32)
```

```
>>> str(vmc_digest("", 24))
'z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXc'
```

```
>>> vmc_digest("", 17)
Traceback (most recent call last):
...
ValueError: digest_size must be a multiple of 3
```

```
>>> vmc_digest("", 66)
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
ValueError: digest_size must be between 0 and 63 (bytes)
```

SHA-512 is 2x faster than SHA1 on modern 64-bit platforms. However, few appliations require 512 bits (64 bytes) of keyspace. That larger size translates into proportionally larger key size requirements, with attendant performance implications. By truncating the SHA-512 digest [1], users may obtain a tunable level of collision avoidance.

The string returned by this function is Base 64 encoded with URL-safe characters [2], making it suitable for use with URLs or filesystem paths. Base 64 encoding results in an output string that is 4/3 the size of the input. If the length of the input string is not divisible by 3, the output is right-padded with equal signs (=), which have no information content. Therefore, this function requires that digest_size is evenly divisible by 3. (The resulting vmc_digest will be 4/3*digest_size bytes.)

According to [3], the probability of a collision using b bits with m messages (sequences) is:

```
P(b, m) = m^2 / 2^(b+1).
```

Note that the collision probability depends on the number of messages, but not their size. Solving for the number of messages:

```
m(b, P) = sqrt(P * 2^(b+1))
```

Solving for the number of *bits*:

```
b(m, P) = log2(m^2/P) - 1
```

For various values of `m` and `P`, the number of *bytes* required according to `b(m,P)` rounded to next multiple of 3 is:

#m	P<1e-24	P<1e-21	P<1e-18	P<1e-15	P<1e-12	P<1e-09
1e+06	15	12	12	9	9	9
1e+09	15	15	12	12	9	9
1e+12	15	15	15	12	12	9
1e+15	18	15	15	15	12	12
1e+18	18	18	15	15	15	12
1e+21	21	18	18	15	15	15
1e+24	21	21	18	18	15	15

For example, given `1e+18` expected messages and a desired collision probability `< 1e-15`, we use `digest_size = 15` (bytes).

References

- [1] http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
- [2] https://tools.ietf.org/html/rfc3548#section-4
- [3] http://stackoverflow.com/a/4014407/342839
- [4] http://stackoverflow.com/a/22029380/342839
- [5] http://preshing.com/20110504/hash-collision-probabilities/
- [6] https://en.wikipedia.org/wiki/Birthday_problem

1.2 Module contents

CHAPTER

TWO

LICENSE

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
```

(continues on next page)

(continued from previous page)

```
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
```

(continues on next page)

(continued from previous page)

```
        attribution notices from the Source form of the Work,
        excluding those notices that do not pertain to any part of
        the Derivative Works; and

    (d) If the Work includes a "NOTICE" text file as part of its
        distribution, then any Derivative Works that You distribute must
        include a readable copy of the attribution notices contained
        within such NOTICE file, excluding those notices that do not
        pertain to any part of the Derivative Works, in at least one
        of the following places: within a NOTICE text file distributed
        as part of the Derivative Works; within the Source form or
        documentation, if provided along with the Derivative Works; or,
        within a display generated by the Derivative Works, if and
        wherever such third-party notices normally appear. The contents
        of the NOTICE file are for informational purposes only and
        do not modify the License. You may add Your own attribution
        notices within Derivative Works that You distribute, alongside
        or as an addendum to the NOTICE text from the Work, provided
        that such additional attribution notices cannot be construed
        as modifying the License.

    You may add Your own copyright statement to Your modifications and
    may provide additional or different license terms and conditions
    for use, reproduction, or distribution of Your modifications, or
    for any such Derivative Works as a whole, provided Your use,
    reproduction, and distribution of the Work otherwise complies with
    the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
    any Contribution intentionally submitted for inclusion in the Work
    by You to the Licensor shall be under the terms and conditions of
    this License, without any additional terms or conditions.
    Notwithstanding the above, nothing herein shall supersede or modify
    the terms of any separate license agreement you may have executed
    with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
    names, trademarks, service marks, or product names of the Licensor,
    except as required for reasonable and customary use in describing the
    origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
    agreed to in writing, Licensor provides the Work (and each
    Contributor provides its Contributions) on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
    implied, including, without limitation, any warranties or conditions
    of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
    PARTICULAR PURPOSE. You are solely responsible for determining the
    appropriateness of using or redistributing the Work and assume any
    risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
    whether in tort (including negligence), contract, or otherwise,
    unless required by applicable law (such as deliberate and grossly
    negligent acts) or agreed to in writing, shall any Contributor be
    liable to You for damages, including any direct, indirect, special,
    incidental, or consequential damages of any character arising as a
```

(continues on next page)

(continued from previous page)

```
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "{}"
      replaced with your own identifying information. (Don't include
      the brackets!)  The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright 2019 bioutils Contributors

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```

CHAPTER

THREE

AUTHORS

Reece Hart <reecehart@gmail.com>
Alan Rubin <alan.rubin@wehi.edu.au>
Andreas Prlic <andreas.prlic@invitae.com>
Timothy Laurent <timothyjlaurent@gmail.com>
Ben Robinson <ben.robinson@invitae.com>
Lucas Wiman <lucas.wiman@gmail.com>

CHAPTER

FOUR

CHANGE LOG

4.1 0.4 Series

4.1.1 0.4.4 (2019-05-13)

Changes since 0.4.3 (2019-04-05).

Special Attention

- This is the last release in the 0.4 series.

Future biocommons packages will be tested and supported only on Python >= 3.6 (https://github.com/biocommons/org/wiki/Migrating-to-Python-3.6)

4.1.2 0.4.3 (2019-04-05)

Changes since 0.4.2 (2019-02-21).

New Features

- Fixes #16: Retry seqfetcher when rate limit exceeded [92d7210]

4.1.3 0.4.2 (2019-02-21)

Changes since 0.4.1 (2019-02-21).

Internal and Developer Changes

- reraise all requests exceptions (not just HTTPError) as RuntimeError [daece64]

4.1.4 0.4.1 (2019-02-21)

Changes since 0.4.0 (2018-11-11).

Other Changes

- expose underlying exception on http failure [9e56110]

Internal and Developer Changes

- updated badges [8f91ed1]
- added LICENSE [b3d6d64]
- added missing contributors definition [97f78b3]
- updated badge list [de2bf15]
- sync'd project files with eutils [3102695]

4.1.5 0.4.0 (2018-10-22)

Changes since 0.3.3 (2017-09-03).

Important Notice

Support for Python <3.6 will be dropped on 2019-03-31. See https://github.com/biocommons/org/wiki/Migrating-to-Python-3.6

New Features

- Closes #10: Support NCBI API keys (and NCBI_API_KEY env variable) [8739c98] (@timothyjlaurent)
- Closes #12: add infer_namespaces and infer_namespace functions [2a53c7f]
- Dropped biopython dependency [0382b86] (@afrubin)
- Added bioutils.sequences.py:elide_sequence() function [018a762]
- Added GRCh38.p12 [3876f36]

4.2 0.5 Series

4.2.1 0.5.5 (2021-05-03)

Changes since 0.5.4 (2021-05-02).

Bug Fixes

- Don't retry sequence fetch with invalid coordinates [94e80cd] (pjcoenen)

4.2.2 0.5.4 (2021-05-02)

Changes since 0.5.3 (2021-04-14).

Internal and Developer Changes

- #31: improve support for degenerate codons [ebcec67] (kayleeyuhas)

4.2.3 0.5.3 (2021-04-14)

Changes since 0.5.2 (2019-11-06).

New Features

- #29: Support ambiguity codes in translation [669a653] (kayleeyuhas)
- added bin/fasta-ga4gh-identifier [63d1078] (Reece Hart)

Internal and Developer Changes

- updated Makefile for Python 3.8 [29eecf5] (Reece Hart)
- fix failing test and reformat [7cc5ebb] (kayleeyuhas)
- improve variable names and use string instead of list [5d7484b] (kayleeyuhas)

4.2.4 0.5.2 (2019-11-06)

Changes since 0.5.1 (2019-07-31).

Special Attention

- Thanks to @trentwatt for significant documentation contributions! See

https://bioutils.readthedocs.io/en/master/ for his handiwork.

Other Changes

- Added changelogs for 0.5.0 and 0.5.1, which @reece forgot to include :-(
- #22 added function docs for all modules [c0090ed] (trentwatt)
- #23: fix setup.cfg description tags (*description* → *long-description*) [8945c04] (Reece Hart)

4.2.5 0.5.1 (2019-07-31)

Changes since 0.5.0 (2019-07-22).

Internal and Developer Changes

- Closes #26: Fix LICENSE filename typo that prevented wheel builds :-([df2fe4a] (Reece Hart)

4.2.6 0.5.0 (2019-07-22)

Changes since 0.4.4 (2019-05-13).

Special Attention

- All biocommons packages now require Python >= 3.6. See https://github.com/biocommons/org/wiki/Migrating-to-Python-3.6

New Features

- #18: Implemented comprehensive sequence normalization (trim, left, right, expand/voca, vcf) [36785fa] (Reece Hart)
- #20: implement hex-based digests à la refget [140a20e] (Reece Hart)
- Add support for cytobands, incl data files from UCSC [0ba4361] (Reece Hart)
- Added accessions.py:coerce_namespace() [e31e592] (Reece Hart)

Internal and Developer Changes

- Added pytest-optional-tests; use test alias in Makefile [ba9b993] (Reece Hart)
- Added trinuc normalization tests [cfe3a68] (Reece Hart)
- Added vcrpy to test requirements [95893f1] (Reece Hart)
- Moved source to src/; updated setup.cfg [ff45fb0] (Reece Hart)
- Removed pip install from tox in favor of deps [8c8f91a] (Reece Hart)
- Renamed doc → docs [1612e5c] (Reece Hart)
- Store assemblies as compressed json [ea79e71] (Reece Hart)
- Update tests to use new vcr cassettes on optional tests (much faster!) [2001745] (Reece Hart)

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX