

# Pylogeny

## API Documentation

February 26, 2015

## Contents

<b>Contents</b>	<b>1</b>
<b>1 Package pylogeny</b>	<b>2</b>
1.1 Modules . . . . .	2
<b>2 Module pylogeny.JSONWriter</b>	<b>3</b>
2.1 Variables . . . . .	3
2.2 Class JSONWriter . . . . .	3
2.2.1 Methods . . . . .	3
2.2.2 Properties . . . . .	3
<b>3 Module pylogeny.__version__</b>	<b>5</b>
3.1 Variables . . . . .	5
<b>4 Module pylogeny.alignment</b>	<b>6</b>
4.1 Variables . . . . .	6
4.2 Class alignment . . . . .	6
4.2.1 Methods . . . . .	6
4.2.2 Properties . . . . .	8
4.3 Class phylipeFriendlyAlignment . . . . .	8
4.3.1 Methods . . . . .	9
4.3.2 Properties . . . . .	10
<b>5 Module pylogeny.base</b>	<b>11</b>
5.1 Functions . . . . .	11
5.2 Variables . . . . .	11
5.3 Class treeStructure . . . . .	11
5.3.1 Methods . . . . .	11
5.3.2 Properties . . . . .	12
5.3.3 Class Variables . . . . .	12
5.4 Class treeNode . . . . .	13
5.4.1 Methods . . . . .	13
5.4.2 Properties . . . . .	13
5.4.3 Class Variables . . . . .	14
5.5 Class treeBranch . . . . .	14
5.5.1 Methods . . . . .	14
5.5.2 Properties . . . . .	14
5.5.3 Class Variables . . . . .	15

5.6	Class trieNode . . . . .	15
5.6.1	Methods . . . . .	15
5.6.2	Properties . . . . .	16
5.6.3	Class Variables . . . . .	16
5.7	Class trie . . . . .	16
5.7.1	Methods . . . . .	17
5.7.2	Properties . . . . .	18
5.7.3	Class Variables . . . . .	18
5.8	Class patriciaTree . . . . .	18
5.8.1	Methods . . . . .	19
5.8.2	Properties . . . . .	19
5.8.3	Class Variables . . . . .	20
<b>6</b>	<b>Module pylogeny.database</b>	<b>21</b>
6.1	Variables . . . . .	21
6.2	Class DatabaseLandscape . . . . .	21
6.2.1	Methods . . . . .	21
6.2.2	Properties . . . . .	22
6.2.3	Class Variables . . . . .	22
6.3	Class SQLExhaustiveLandscape . . . . .	23
6.3.1	Methods . . . . .	23
6.3.2	Properties . . . . .	25
6.3.3	Class Variables . . . . .	25
6.4	Class SQLiteLandscape . . . . .	26
6.4.1	Methods . . . . .	26
6.4.2	Properties . . . . .	27
6.4.3	Class Variables . . . . .	27
6.5	Class database . . . . .	27
6.5.1	Methods . . . . .	28
6.5.2	Properties . . . . .	29
6.5.3	Class Variables . . . . .	29
6.6	Class SQLDatabase . . . . .	29
6.6.1	Methods . . . . .	29
6.6.2	Properties . . . . .	30
6.6.3	Class Variables . . . . .	30
6.7	Class SQLiteDatabase . . . . .	31
6.7.1	Methods . . . . .	31
6.7.2	Properties . . . . .	32
6.7.3	Class Variables . . . . .	32
<b>7</b>	<b>Module pylogeny.executable</b>	<b>33</b>
7.1	Functions . . . . .	33
7.2	Variables . . . . .	33
7.3	Class aTemporaryDirectory . . . . .	33
7.3.1	Methods . . . . .	33
7.3.2	Properties . . . . .	34
7.4	Class executable . . . . .	34
7.4.1	Methods . . . . .	34
7.4.2	Properties . . . . .	34
7.4.3	Class Variables . . . . .	35
7.5	Class treepuzzle . . . . .	35
7.5.1	Methods . . . . .	35

7.5.2	Properties . . . . .	36
7.5.3	Class Variables . . . . .	36
7.6	Class <code>consel</code> . . . . .	36
7.6.1	Methods . . . . .	36
7.6.2	Properties . . . . .	37
7.6.3	Class Variables . . . . .	37
7.7	Class <code>fasttree</code> . . . . .	37
7.7.1	Methods . . . . .	37
7.7.2	Properties . . . . .	38
7.7.3	Class Variables . . . . .	38
7.8	Class <code>raxml</code> . . . . .	38
7.8.1	Methods . . . . .	39
7.8.2	Properties . . . . .	39
7.8.3	Class Variables . . . . .	39
7.9	Class <code>rspr</code> . . . . .	40
7.9.1	Methods . . . . .	40
7.9.2	Properties . . . . .	40
7.9.3	Class Variables . . . . .	40
<b>8</b>	<b>Module <code>pylogeny.heuristic</code></b>	<b>42</b>
8.1	Variables . . . . .	42
8.2	Class <code>heuristic</code> . . . . .	42
8.2.1	Methods . . . . .	42
8.2.2	Properties . . . . .	43
8.3	Class <code>phylogeneticLinearHeuristic</code> . . . . .	43
8.3.1	Methods . . . . .	43
8.3.2	Properties . . . . .	43
8.3.3	Class Variables . . . . .	44
8.4	Class <code>parsimonyGreedy</code> . . . . .	44
8.4.1	Methods . . . . .	44
8.4.2	Properties . . . . .	45
8.4.3	Class Variables . . . . .	45
8.5	Class <code>likelihoodGreedy</code> . . . . .	45
8.5.1	Methods . . . . .	45
8.5.2	Properties . . . . .	46
8.5.3	Class Variables . . . . .	46
8.6	Class <code>smoothGreedy</code> . . . . .	46
8.6.1	Methods . . . . .	47
8.6.2	Properties . . . . .	47
8.6.3	Class Variables . . . . .	47
8.7	Class <code>RAxMLIdentify</code> . . . . .	48
8.7.1	Methods . . . . .	48
8.7.2	Properties . . . . .	49
8.7.3	Class Variables . . . . .	49
<b>9</b>	<b>Module <code>pylogeny.landscape</code></b>	<b>50</b>
9.1	Variables . . . . .	50
9.2	Class <code>graph</code> . . . . .	50
9.2.1	Methods . . . . .	50
9.2.2	Properties . . . . .	53
9.3	Class <code>landscape</code> . . . . .	53
9.3.1	Methods . . . . .	53

9.3.2	Properties . . . . .	58
9.3.3	Class Variables . . . . .	58
9.4	Class vertex . . . . .	58
9.4.1	Methods . . . . .	59
9.4.2	Properties . . . . .	61
<b>10</b>	<b>Module pylogeny.landscapeWriter</b>	<b>62</b>
10.1	Variables . . . . .	62
10.2	Class landscapeWriter . . . . .	62
10.2.1	Methods . . . . .	62
10.2.2	Properties . . . . .	62
10.3	Class landscapeParser . . . . .	63
10.3.1	Methods . . . . .	63
10.3.2	Properties . . . . .	63
<b>11</b>	<b>Module pylogeny.model</b>	<b>64</b>
11.1	Variables . . . . .	64
11.2	Class PhyloModelError . . . . .	64
11.2.1	Methods . . . . .	64
11.2.2	Properties . . . . .	65
11.3	Class DiscreteStateModel . . . . .	65
11.3.1	Methods . . . . .	65
11.3.2	Properties . . . . .	66
<b>12</b>	<b>Module pylogeny.newick</b>	<b>67</b>
12.1	Functions . . . . .	67
12.2	Variables . . . . .	68
12.3	Class ParsingError . . . . .	68
12.3.1	Methods . . . . .	68
12.3.2	Properties . . . . .	69
12.4	Class node . . . . .	69
12.4.1	Methods . . . . .	69
12.4.2	Properties . . . . .	70
12.4.3	Class Variables . . . . .	70
12.5	Class branch . . . . .	70
12.5.1	Methods . . . . .	70
12.5.2	Properties . . . . .	71
12.5.3	Class Variables . . . . .	71
12.6	Class newickParser . . . . .	71
12.6.1	Methods . . . . .	71
<b>13</b>	<b>Module pylogeny.parsimony</b>	<b>72</b>
13.1	Functions . . . . .	72
13.2	Variables . . . . .	72
13.3	Class profile_set . . . . .	72
13.3.1	Methods . . . . .	72
13.4	Class site_profile . . . . .	73
13.4.1	Methods . . . . .	73
<b>14</b>	<b>Module pylogeny.pll</b>	<b>74</b>
14.1	Variables . . . . .	74
14.2	Class dataModel . . . . .	74

14.2.1	Methods . . . . .	74
14.3	Class partitionModel . . . . .	74
14.3.1	Methods . . . . .	75
<b>15</b>	<b>Module pylogeny.rearrangement</b>	<b>76</b>
15.1	Functions . . . . .	76
15.2	Variables . . . . .	76
15.3	Class RearrangementError . . . . .	76
15.3.1	Methods . . . . .	76
15.3.2	Properties . . . . .	77
15.4	Class rearrangement . . . . .	77
15.4.1	Methods . . . . .	77
15.5	Class topology . . . . .	78
15.5.1	Methods . . . . .	78
15.5.2	Properties . . . . .	81
15.5.3	Class Variables . . . . .	82
<b>16</b>	<b>Module pylogeny.scoring</b>	<b>83</b>
16.1	Functions . . . . .	83
16.2	Variables . . . . .	84
<b>17</b>	<b>Module pylogeny.tree</b>	<b>85</b>
17.1	Functions . . . . .	85
17.2	Variables . . . . .	85
17.3	Class tree . . . . .	85
17.3.1	Methods . . . . .	85
17.3.2	Properties . . . . .	87
17.4	Class treeSet . . . . .	87
17.4.1	Methods . . . . .	87
17.4.2	Properties . . . . .	89
17.4.3	Class Variables . . . . .	89
17.5	Class bipartition . . . . .	89
17.5.1	Methods . . . . .	89
17.5.2	Properties . . . . .	91

# 1 Package pylogeny

Pylogeny is a Python library and code framework for phylogenetic tree reconstruction and scoring.

Allows one to perform the following tasks: (1) Generate and manage phylogenetic tree landscapes. (2) Build and rearrange phylogenetic trees using preset operators such as NNI, SPR, and TBR. (3) Score phylogenetic trees by Log-likelihood and Parsimony.

Dependencies: Pandas, P4 Phylogenetic Library. Suggested: FastTree, RAxML, PytBEAGLEhon.

## 1.1 Modules

- **JSONWriter**: Serialize a phylogenetic landscape into a JSON object.  
(Section 2, p. 3)
- **\_\_version\_\_** (Section 3, p. 5)
- **alignment**: Handle input biological sequence alignment files for the purposes of phylogenetic inference.  
(Section 4, p. 6)
- **base**: Definitions for generalized containers and objects used by other structures in this framework.  
(Section 5, p. 11)
- **database**: Connect, access, + manipulate external tree data from a remote SQL server or from a sqlite file.  
(Section 6, p. 21)
- **executable**: Defines an interface to manage interfacing with the system for respective application calls and implements some of these for executables such as FastTree and RAxML.  
(Section 7, p. 33)
- **heuristic**: Define the interface for a heuristic in order to implement any manner of heuristic for a combinatorial problem that can be abstracted into a state graph.  
(Section 8, p. 42)
- **landscape**: Encapsulate a phylogenetic tree space.  
(Section 9, p. 50)
- **landscapeWriter**: Serialize a phylogenetic landscape into an SQLite database file made up of three components: all tree IDs and respective scores, the alignment file as a set of sequences, and a representation of the graph as an edge list.  
(Section 10, p. 62)
- **model**: Phylogenetic tree scoring models; intended to be coupled with the use of pytbeaglehon (BEAGLE) high-performance library.  
(Section 11, p. 64)
- **newick**: Newick string parsing and object interaction.  
(Section 12, p. 67)
- **parsimony**: Toolkit for performance of Parsimonious Criterion (Parsimony) methods of optimization of a phylogenetic topology with a particular set of data.  
(Section 13, p. 72)
- **pll**: Wrap C extension for libpll library for use in natural Python.  
(Section 14, p. 74)
- **rearrangement**: Phylogenetic tree structure encapsulation; allow rearrangement of said structure.  
(Section 15, p. 76)
- **scoring**: Functions for phylogenetic tree goodness-of-fit scoring.  
(Section 16, p. 83)
- **tree**: Container definition for (phylogenetic) bifurcating or multifurcating trees defined using Newick strings, collections of them, and for splits of these trees.  
(Section 17, p. 85)

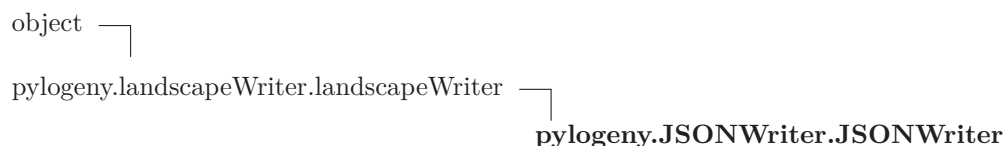
## 2 Module *pylogeny.JSONWriter*

Serialize a phylogenetic landscape into a JSON object.

### 2.1 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> 'pylogeny'

### 2.2 Class *JSONWriter*



#### 2.2.1 Methods

`__init__(self, ls, name)`

x.`__init__`(...) initializes x; see `help(type(x))` for signature

Overrides: object.`__init__` `__init__` `__init__` (inherited documentation)

`nodeToJSON(self, node)`

`getOnlyImprovements(self, groups=None)`

`getCompleteLandscape(self)`

`getJSON(self)`

*Inherited from `pylogeny.landscapeWriter.landscapeWriter` (Section 10.2)*

`writeFile()`

*Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

#### 2.2.2 Properties

Name	Description
<i>Inherited from object</i> __class__	



### 3 Module *pylogeny.\_\_version\_\_*

#### 3.1 Variables

Name	Description
VERSION	<b>Value:</b> '0.3.4.0'
__package__	<b>Value:</b> None

## 4 Module pylogeny.alignment

Handle input biological sequence alignment files for the purposes of phylogenetic inference. Will read all types of alignment files by utilizing the P4 python phylogenetic library.

### 4.1 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> 'pylogeny'

### 4.2 Class alignment

object └─  
           **pylogeny.alignment.alignment**

**Known Subclasses:** pylogeny.alignment.phylipFriendlyAlignment

Wrap a biological sequence alignment to enable functionality necessary for phylogenetic inference. Makes use of temporary files; requires to be closed once no longer needed.

#### 4.2.1 Methods

<b><code>__init__</code></b> ( <i>self</i> , <i>inal=None</i> ) Instantiate an object intended to wrap an alignment for the purposes of running phylogenetic inference. :param inal: An alignment file path (most formats are accepted). Overrides: object. <code>__init__</code>
<b><code>__getitem__</code></b> ( <i>self</i> , <i>i</i> )
<b><code>__str__</code></b> ( <i>self</i> ) <code>str(x)</code> Overrides: object. <code>__str__</code> extit(inherited documentation)
<b><code>__len__</code></b> ( <i>self</i> )

<b>__iter__</b> ( <i>self</i> )
<b>close</b> ( <i>self</i> ) Delete all temporary files and clear data.
<b>toStrList</b> ( <i>self</i> ) Get all sequences as a list of strings.
<b>getStateModel</b> ( <i>self</i> )
<b>getSize</b> ( <i>self</i> ) Return the size of the alignment, or how many characters there are in each respective item in the alignment.
<b>getNumSeqs</b> ( <i>self</i> ) Return the number of sequences that are present in the sequence alignment.
<b>getDim</b> ( <i>self</i> ) Return the dimensionality of the sequence alignment (how many different types of characters).
<b>getSequence</b> ( <i>self</i> , <i>i</i> ) Acquire the <i>i</i> th sequence.
<b>getFASTA</b> ( <i>self</i> ) Get (and create if not already) a path to a temporary FASTA file. This will be deleted upon closure of the alignment instance.
<b>getApproxMLNewick</b> ( <i>self</i> ) Get a tree in newick format via use of FastTree that serves as an approximation of the maximum likelihood tree for this data.
<b>getApproxMLTree</b> ( <i>self</i> ) Get a tree object for an approximation of the maximum likelihood tree for this data using FastTree.

<b>getFastTreeNewick</b> ( <i>self</i> )
--

Alias for the "getApproxMLNewick()" function.
---

<b>getTaxa</b> ( <i>self</i> )
--------------------------------

Return taxa names.
--------------------

<b>getAlignment</b> ( <i>self</i> )
-------------------------------------

Acquire the alignment data structure (P4 module).
---

<b>bootstrap</b> ( <i>self</i> )
----------------------------------

Perform bootstrapping on the alignment data.
--

### *Inherited from object*

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(),  
 \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_(),  
 \_\_subclasshook\_\_()

#### 4.2.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

### 4.3 Class *phylipFriendlyAlignment*

object └─

pylogeny.alignment.alignment └─

**pylogeny.alignment.phylipFriendlyAlignment**

An alignment object that renames all comprising taxa in order to be able to be written as a strict Phylip file.

## 4.3.1 Methods

**\_\_init\_\_**(*self*, *inal*=None)

Instantiate an object intended to wrap an alignment for the purposes of running phylogenetic inference.

:param *inal*: An alignment file path (most formats are accepted).

Overrides: object.\_\_init\_\_ *exitit*(inherited documentation)

**getPhylip**(*self*)

Get a path to a temporary Phylip file. This will be deleted upon closure of the alignment instance.

**writeProperNexus**(*self*, *wri*)

Write a Nexus file with proper names.

**reassignFromReinterpretedNewick**(*self*, *tr*)

Replace all proper names with reassigned names in a Newick tree.

**reinterpretNewick**(*self*, *tr*)

Replaces all reassigned names to proper names in a Newick tree.

**getProperName**(*self*, *n*)

Return the actual name for an integer-based sequence name that was reassigned at initialization.

**getTaxa**(*self*)

Return current taxa names in the alignment.

Overrides: *pylogeny.alignment.alignment.getTaxa*

**recreateObject**(*self*)

Reintializes the object.

*Inherited from **pylogeny.alignment.alignment**(Section 4.2)*

*\_\_getitem\_\_*(), *\_\_iter\_\_*(), *\_\_len\_\_*(), *\_\_str\_\_*(), *bootstrap*(), *close*(), *getAlignment*(), *getApproxMLNewick*(), *getApproxMLTree*(), *getDim*(), *getFASTA*(), *getFastTreeNewick*(), *getNumSeqs*(), *getSequence*(), *getSize*(), *getStateModel*(),

toStrList()

***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__subclasshook__()`

**4.3.2 Properties**

Name	Description
<i>Inherited from object</i> <code>__class__</code>	

## 5 Module *pylogeny.base*

Definitions for generalized containers and objects used by other structures in this framework.

### 5.1 Functions

<b>longest_common_substring</b> ( <i>s1</i> , <i>s2</i> )
Simplified, traditional LCS algorithm implementation.

### 5.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> 'pylogeny'

### 5.3 Class *treeStructure*



**Known Subclasses:** *pylogeny.base.trie*, *pylogeny.rearrangement.topology*

Defines a base collection of *treeNodes* and *treeBranches* in a hierarchical tree structure.

#### 5.3.1 Methods

<b>__init__</b> ( <i>self</i> , <i>root</i> =None)
<i>x</i> . <b>__init__</b> (...) initializes <i>x</i> ; see <code>help(type(x))</code> for signature
Overrides: <i>object</i> . <b>__init__</b> <i>extit</i> (inherited documentation)

<b>__contains__</b> ( <i>self</i> , <i>x</i> )
Determines whether a node is found in the tree structure.
Overrides: <i>_abcoll.Container</i> . <b>__contains__</b>

<b>getRoot</b> ( <i>self</i> )
--------------------------------

Return the top-level, root, node of the tree.
---

<b>leaves</b> ( <i>root</i> )
-------------------------------

<b>getAllLeaves</b> ( <i>self</i> )
-------------------------------------

<b>nodes</b> ( <i>root</i> )
------------------------------

<b>getAllNodes</b> ( <i>self</i> )
------------------------------------

<b>postOrderTraversal</b> ( <i>root</i> )
---

<b>getPostOrderTraversal</b> ( <i>self</i> )
--

<b>__str__</b> ( <i>self</i> )
--------------------------------

Returns a string representation of the tree.
--

Overrides: object.__str__
---------------------------

*Inherited from* ***\_\_abcoll.Container***

<b>__subclasshook__</b> ()
----------------------------

*Inherited from* ***object***

<b>__delattr__</b> ()	<b>__format__</b> ()	<b>__getattr__</b> ()	<b>__hash__</b> ()	<b>__new__</b> ()
<b>__reduce__</b> ()	<b>__reduce_ex__</b> ()	<b>__repr__</b> ()	<b>__setattr__</b> ()	<b>__sizeof__</b> ()

### 5.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<b>__class__</b>	

### 5.3.3 Class Variables

Name	Description
<b>root</b>	<b>Value:</b> None
<b>__abstractmethods__</b>	<b>Value:</b> frozenset([])



## 5.4 Class `treeNode`



**Known Subclasses:** `pylogeny.base.trieNode`, `pylogeny.newick.node`

A node in a tree.

### 5.4.1 Methods

```
__init__(self, lbl=None, children=None, parent=None)
```

`x.__init__(...)` initializes `x`; see `help(type(x))` for signature

Overrides: `object.__init__` `extit`(inherited documentation)

```
getLabel(self)
```

```
getParent(self)
```

```
addChild(self, item)
```

```
getChildByIndex(self, i)
```

```
getChildren(self)
```

```
isLeaf(self)
```

```
isInternalNode(self)
```

*Inherited from `object`*

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),
__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),
__str__(), __subclasshook__()
```

### 5.4.2 Properties

Name	Description
<i>Inherited from <code>object</code></i>	
<code>__class__</code>	

### 5.4.3 Class Variables

Name	Description
label	<b>Value:</b> None
parent	<b>Value:</b> None
children	<b>Value:</b> None

## 5.5 Class *treeBranch*

object —  
     **pylogeny.base.treeBranch**

**Known Subclasses:** *pylogeny.newick.branch*

A branch in a tree.

### 5.5.1 Methods

```
__init__(self, parent=None, child=None, label='')
```

x.**\_\_init\_\_**(...) initializes x; see help(type(x)) for signature

Overrides: object.**\_\_init\_\_** extit(inherited documentation)

```
getLabel(self)
```

```
getParent(self)
```

```
getChild(self)
```

*Inherited from object*

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),  

__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),  

__str__(), __subclasshook__()
```

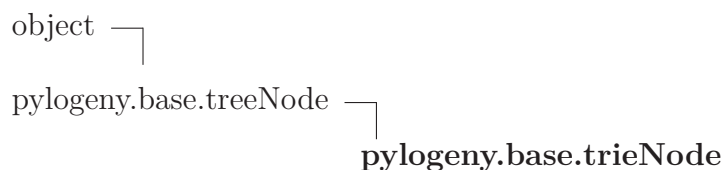
### 5.5.2 Properties

Name	Description
<i>Inherited from object</i> <b>__class__</b>	

### 5.5.3 Class Variables

Name	Description
label	<b>Value:</b> ''
parent	<b>Value:</b> None
child	<b>Value:</b> None

## 5.6 Class *trieNode*



A subclass of *treeNode* that allows for checking non-zero members amongst children branches and other conveniences.

### 5.6.1 Methods

<b>getParentNode</b> ( <i>self</i> )
--------------------------------------

<b>setChildNode</b> ( <i>self</i> , <i>child</i> , <i>newchild</i> )
--

<b>iterNonEmptyChildrenNodes</b> ( <i>self</i> )
--

Iterate over all children that are not empty.
---

<b>getNonEmptyChildrenNodes</b> ( <i>self</i> )
---

Acquire a list of all non-empty children.
---

<b>getNonEmptyChildrenBranches</b> ( <i>self</i> )
--

Acquire a list of all non-empty children.
---

<b>getNonEmptyChildrenBranchLabels</b> ( <i>self</i> )
--

<b>numEmptyChildrenNodes</b> ( <i>self</i> )
--

Acquire the number of children nodes that are marked 0 or nonexistent.
--

*Inherited from pylogeny.base.treeNode(Section 5.4)*

`__init__()`, `addChild()`, `getChildByIndex()`, `getChildren()`, `getLabel()`, `getParent()`, `isInternalNode()`, `isLeaf()`

### *Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

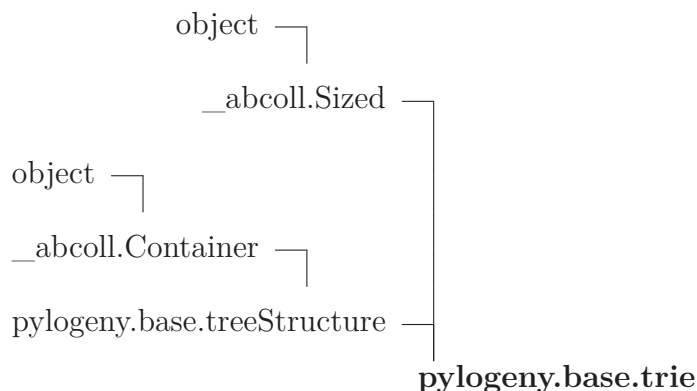
#### 5.6.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

#### 5.6.3 Class Variables

Name	Description
<i>Inherited from <code>pylogeny.base.treeNode</code> (Section 5.4)</i>	
<code>children</code> , <code>label</code> , <code>parent</code>	

### 5.7 Class trie



**Known Subclasses:** `pylogeny.base.patriciaTree`

Defines a trie across a range of strings.

**5.7.1 Methods**

**`__init__`**(*self*)

*x*.`__init__`(...) initializes *x*; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

**`__contains__`**(*self*, *x*)

Implementing for interface (Container).

Overrides: `__abcoll.Container.__contains__`

**`__len__`**(*self*)

Implementing for interface (Sized).

Overrides: `__abcoll.Sized.__len__`

**`getAlphabet`**(*self*)

**`getRoot`**(*self*)

Return the top-level, root, node of the tree.

Overrides: `pylogeny.base.treeStructure.getRoot` `exitit`(inherited documentation)

**`search`**(*self*, *seq*)

Search for a sequence in the trie. Returns true if it exists.

**`insert`**(*self*, *seq*)

Dynamically insert a sequence into the trie.

**`delete`**(*self*, *seq*)

Remove a sequence from the trie. Will not remove added characters to alphabet.

*Inherited from* `__abcoll.Sized`

`__subclasshook__`()

*Inherited from* `pylogeny.base.treeStructure`(Section 5.3)

`__str__`(), `getAllLeaves`(), `getAllNodes`(), `getPostOrderTraversal`(), `leaves`(), `nodes`(), `postOrderTraversal`()

***Inherited from object***

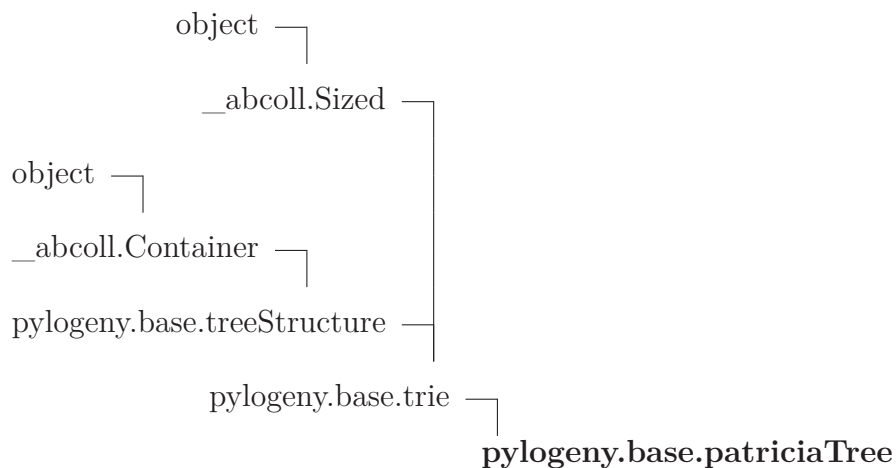
`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`

**5.7.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**5.7.3 Class Variables**

Name	Description
<code>alphabet</code>	<b>Value:</b> None
<code>root</code>	<b>Value:</b> None
<code>count</code>	<b>Value:</b> 0
<code>nextLabel</code>	<b>Value:</b> 1
<code>__abstractmethods__</code>	<b>Value:</b> <code>frozenset([])</code>

**5.8 Class *patriciaTree***

Defines a PATRICIA tree (condensed trie) across a range of strings.

**5.8.1 Methods**

**\_\_contains\_\_**(*self*, *x*)

Implementing for interface (Container).

Overrides: `__abcoll.Container.__contains__`

**search**(*self*, *seq*)

Search for a sequence in the PATRICIA tree. Returns its position in addition sequence if it exists. Else, returns 0.

Overrides: `pylogeny.base.trie.search`

**insert**(*self*, *seq*)

Dynamically insert a sequence into the PATRICIA tree. Returns the unique index in the tree for that string.

Overrides: `pylogeny.base.trie.insert`

**delete**(*self*, *seq*)

Remove a sequence from the PATRICIA tree. Will not remove added characters to alphabet.

Overrides: `pylogeny.base.trie.delete`

**Inherited from *pylogeny.base.trie* (Section 5.7)**

`__init__()`, `__len__()`, `getAlphabet()`, `getRoot()`

**Inherited from *\_\_abcoll.Sized***

`__subclasshook__()`

**Inherited from *pylogeny.base.treeStructure* (Section 5.3)**

`__str__()`, `getAllLeaves()`, `getAllNodes()`, `getPostOrderTraversal()`, `leaves()`, `nodes()`, `postOrderTraversal()`

**Inherited from *object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`

**5.8.2 Properties**

Name	Description
<i>Inherited from object</i> __class__	

### 5.8.3 Class Variables

Name	Description
<i>Inherited from pylogeny.base.trie (Section 5.7)</i> __abstractmethods__, alphabet, count, nextLabel, root	



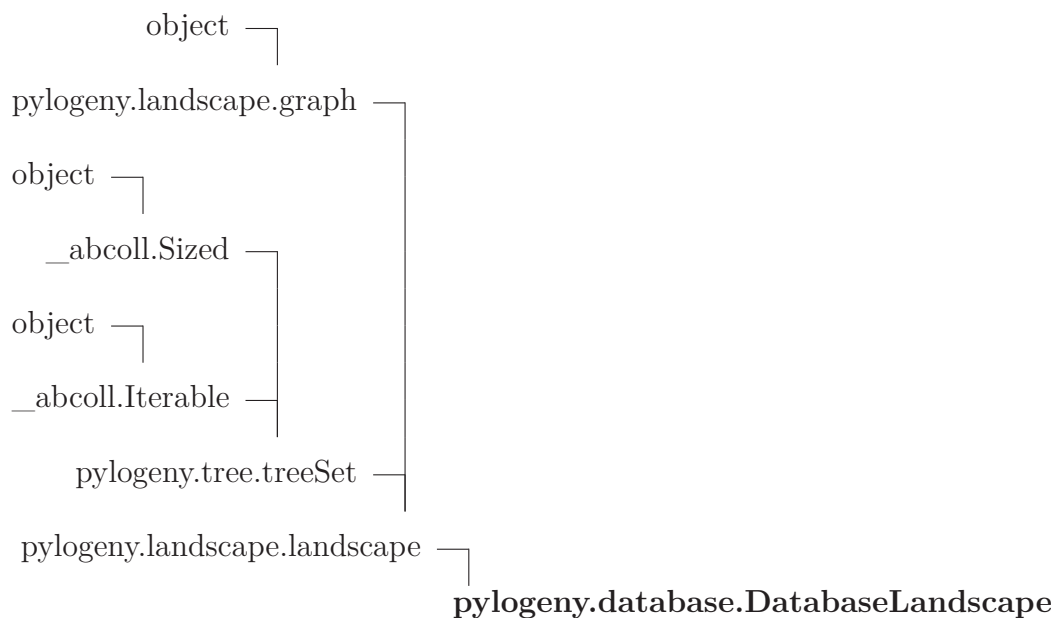
## 6 Module pylogeny.database

Connect, access, + manipulate external tree data from a remote SQL server or from a sqlite file.

### 6.1 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> 'pylogeny'

### 6.2 Class DatabaseLandscape



**Known Subclasses:** `pylogeny.database.SQLExhaustiveLandscape`

Abstract the landscape to one comprising a landscape.

#### 6.2.1 Methods

<b>getNode</b> ( <i>self</i> , <i>i</i> ) Overrides: <code>pylogeny.landscape.graph.getNode</code>
---

*Inherited from `pylogeny.landscape.landscape` (Section 9.3)*

\_\_getitem\_\_(), \_\_init\_\_(), \_\_iter\_\_(), \_\_str\_\_(), addTree(), addTreeByNewick(), exploreRandomTree(), exploreTree(), findTree(), findTreeTopology(), findTreeTopologyByStructure(), getAlignment(), getAllPathsOfBestImprovement(), getBestImprovement(), getBipartitionFoundInTreeByIndex(), getGlobalOptimum(), getLocalOptima(), getLocks(), getNumberTaxa(), getPathOfBestImprovement(), getPossibleNumberRootedTrees(), getPossibleNumberUnrootedTrees(), getRoot(), getRootTree(), getTree(), getVertex(), indexOf(), isLocalOptimum(), isViolating(), iterAllPathsOfBestImprovement(), lockBranchFoundInTree(), lockBranchFoundInTreeByIndex(), removeTree(), removeTreeByIndex(), setAlignment(), setOperator(), toProperNewickTreeSet(), toTreeSet(), toggleLock()

### ***Inherited from pylogeny.landscape.graph(Section 9.2)***

\_\_len\_\_(), clearEdgeWeights(), getCenter(), getCliqueNumber(), getCliques(), getCliquesOfNode(), GetComponentOfNode(), getComponents(), getDegreeFor(), getDiameter(), getEdge(), getEdges(), getEdgesFor(), getMST(), getNeighborsFor(), getNetworkXObject(), getNodeNames(), getNodes(), getNumCliques(), getNumComponents(), getShortestPath(), getShortestPathLength(), getSize(), hasPath(), isEdge(), iterNodes(), setDefaultWeight()

### ***Inherited from pylogeny.tree.treeSet(Section 17.4)***

\_\_setitem\_\_(), fromTreeFile(), iterTrees(), toTreeFile()

### ***Inherited from \_\_abcoll.Sized***

\_\_subclasshook\_\_()

### ***Inherited from object***

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(), \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_()

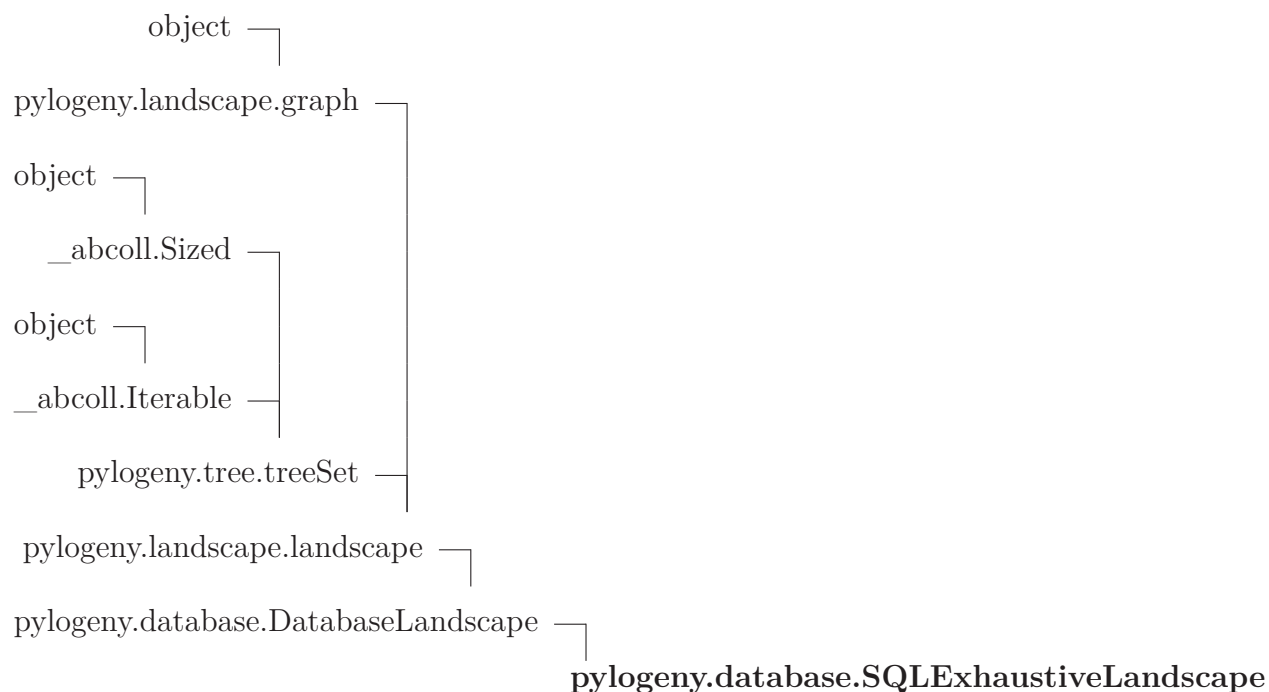
## **6.2.2 Properties**

Name	Description
<i>Inherited from object</i>	
__class__	

## **6.2.3 Class Variables**

Name	Description
__abstractmethods__	<b>Value:</b> frozenset(['_fetchRearrangementsFromDatabase', '_fetchTre...'])

### 6.3 Class *SQLExhaustiveLandscape*



#### 6.3.1 Methods

**\_\_init\_\_**(*self*, *dbobj*, *aliname*)

Initialize the landscape.

:param ali: An :class:'alignment.alignment' object. :param starting\_\_tree: An optional tree object to start the landscape with. :param root: Whether or not to acquire an approximate maximum likelihood tree (FastTree) or start the landscape with a given starting tree. :param operator: A string that describes what operator the landscape is mostly comprised of.

Overrides: *object.\_\_init\_\_* *exitit*(inherited documentation)

**exploreRandomTree(*self*, *i*)**

Acquire a single neighbor to a tree in the landscape by performing a random rearrangement of type SPR (by default), NNI, or TBR – this is done by performing a rearrangement on a random branch in the topology. Rearrangement type is provided as a rearrangement module type definition of form, for example, TYPE\_SPR, TYPE\_NNI, etc.

Overrides: *pylogeny.landscape.landscape.exploreRandomTree* extit(inherited documentation)

**getDatabaseNode(*self*, *i*)****exploreTree(*self*, *i*)**

Get all neighbors to a tree named *i* in the landscape using a respective rearrangement operator as defined in the rearrangement module. Rearrangement type is provided as a rearrangement module type definition of form, for example, TYPE\_SPR, TYPE\_NNI, etc. By default, this is TYPE\_SPR.

Overrides: *pylogeny.landscape.landscape.exploreTree* extit(inherited documentation)

***Inherited from pylogeny.database.DatabaseLandscape(Section 6.2)***

*getNode()*

***Inherited from pylogeny.landscape.landscape(Section 9.3)***

*\_\_getitem\_\_()*, *\_\_iter\_\_()*, *\_\_str\_\_()*, *addTree()*, *addTreeByNewick()*, *findTree()*, *findTreeTopology()*, *findTreeTopologyByStructure()*, *getAlignment()*, *getAllPathsOfBestImprovement()*, *getBestImprovement()*, *getBipartitionFoundInTreeByIndex()*, *getGlobalOptimum()*, *getLocalOptima()*, *getLocks()*, *getNumberTaxa()*, *getPathOfBestImprovement()*, *getPossibleNumberRootedTrees()*, *getPossibleNumberUnrootedTrees()*, *getRoot()*, *getRootTree()*, *getTree()*, *getVertex()*, *indexOf()*, *isLocalOptimum()*, *isViolating()*, *iterAllPathsOfBestImprovement()*, *lockBranchFoundInTree()*, *lockBranchFoundInTreeByIndex()*, *removeTree()*, *removeTreeByIndex()*, *setAlignment()*, *setOperator()*, *toProperNewickTreeSet()*, *toTreeSet()*, *toggleLock()*

***Inherited from pylogeny.landscape.graph(Section 9.2)***

*\_\_len\_\_()*, *clearEdgeWeights()*, *getCenter()*, *getCliqueNumber()*, *getCliques()*, *getCliquesOfNode()*, *getComponentOfNode()*, *getComponents()*, *getDegreeFor()*, *getDiameter()*, *getEdge()*, *getEdges()*, *getEdgesFor()*, *getMST()*, *getNeighborsFor()*, *getNetworkXObject()*, *getNodeNames()*, *getNodes()*, *getNumCliques()*, *getNumComponents()*, *getShortestPath()*, *getShortestPathLength()*, *getSize()*, *has-*

`Path()`, `isEdge()`, `iterNodes()`, `setDefaultWeight()`

***Inherited from `pylogeny.tree.treeSet`(Section 17.4)***

`__setitem__()`, `fromTreeFile()`, `iterTrees()`, `toTreeFile()`

***Inherited from `__abcoll.Sized`***

`__subclasshook__()`

***Inherited from `object`***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`

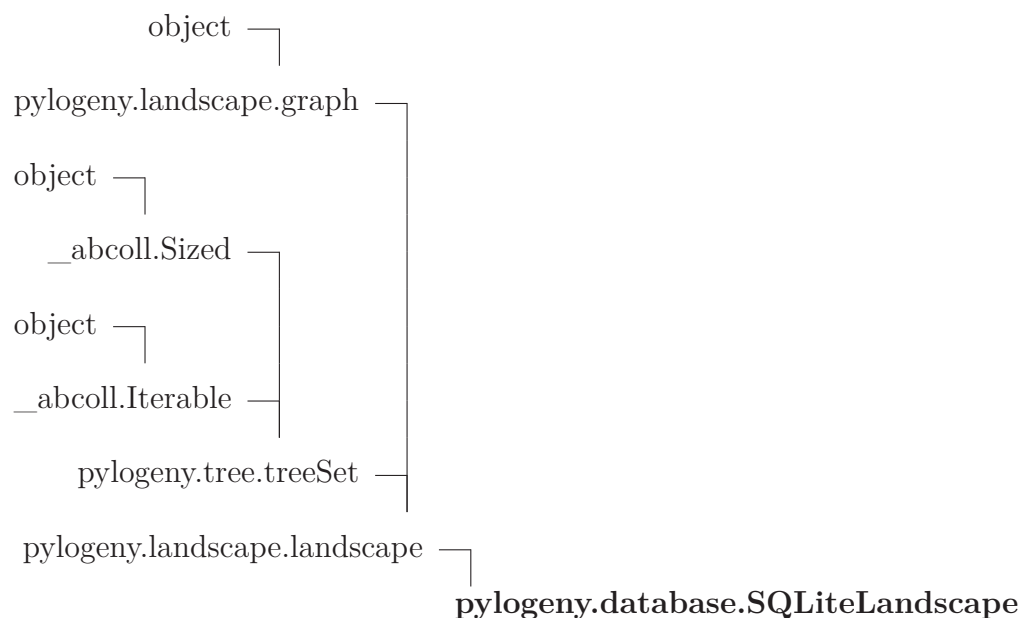
### 6.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 6.3.3 Class Variables

Name	Description
<code>__abstractmethods__</code>	<b>Value:</b> <code>frozenset([])</code>

## 6.4 Class *SQLiteLandscape*



Allow random access of all landscape data from an sqlite file found on the hard disk.

### 6.4.1 Methods

**`__init__(self, dbobj)`**

Initialize the landscape.

:param ali: An :class:'alignment.alignment' object. :param starting\_\_tree: An optional tree object to start the landscape with. :param root: Whether or not to acquire an approximate maximum likelihood tree (FastTree) or start the landscape with a given starting tree. :param operator: A string that describes what operator the landscape is mostly comprised of.

Overrides: object.\_\_init\_\_ exitit(inherited documentation)

***Inherited from `pylogeny.landscape.landscape` (Section 9.3)***

`__getitem__()`, `__iter__()`, `__str__()`, `addTree()`, `addTreeByNewick()`, `exploreRandomTree()`, `exploreTree()`, `findTree()`, `findTreeTopology()`, `findTreeTopologyByStructure()`, `getAlignment()`, `getAllPathsOfBestImprovement()`, `getBestImprovement()`, `getBipartitionFoundInTreeByIndex()`, `getGlobalOptimum()`, `getLocalOptima()`, `getLocks()`, `getNumberTaxa()`, `getPathOfBestImprovement()`, `getPossibleNumberRootedTrees()`, `getPossibleNumberUnrootedTrees()`, `getRoot()`, `getRootTree()`, `getTree()`, `getVertex()`, `indexOf()`, `isLocalOptimum()`, `isViolating()`, `iterAllPath-`

sOfBestImprovement(), lockBranchFoundInTree(), lockBranchFoundInTreeByIndex(), removeTree(), removeTreeByIndex(), setAlignment(), setOperator(), toProperNewickTreeSet(), toTreeSet(), toggleLock()

***Inherited from `pylogeny.landscape.graph`(Section 9.2)***

\_\_len\_\_(), clearEdgeWeights(), getCenter(), getCliqueNumber(), getCliques(), getCliquesOfNode(), getComponentOfNode(), getComponents(), getDegreeFor(), getDiameter(), getEdge(), getEdges(), getEdgesFor(), getMST(), getNeighborsFor(), getNetworkXObject(), getNode(), getNodeNames(), getNodes(), getNumCliques(), getNumComponents(), getShortestPath(), getShortestPathLength(), getSize(), hasPath(), isEdge(), iterNodes(), setDefaultWeight()

***Inherited from `pylogeny.tree.treeSet`(Section 17.4)***

\_\_setitem\_\_(), fromTreeFile(), iterTrees(), toTreeFile()

***Inherited from `__abcoll.Sized`***

\_\_subclasshook\_\_()

***Inherited from `object`***

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(), \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_()

#### 6.4.2 Properties

Name	Description
<i>Inherited from <code>object</code></i>	
__class__	

#### 6.4.3 Class Variables

Name	Description
<i>Inherited from <code>pylogeny.tree.treeSet</code> (Section 17.4)</i>	
__abstractmethods__	

### 6.5 Class database

```

object └─
          pylogeny.database.database

```

**Known Subclasses:** `pylogeny.database.SQLDatabase`, `pylogeny.database.SQLiteDatabase`

Allow interfacing with a SQL/sqlite database.

### 6.5.1 Methods

**getTables**(*self*)

**getColumns**(*self*, *table*)

**isEmpty**(*self*)

Determine if the database is empty.

**getHeaders**(*self*, *table*)

Get only header names for a given table's columns.

**getRecordsColumn**(*self*, *table*, *col*)

Get all data for a single column from records for a table.

**getRecords**(*self*, *table*)

Get all records from a given table in the database.

**iterRecords**(*self*, *table*)

Get a record, one at a time, from a table in the database.

**filterRecords**(*self*, *table*, *condn*)

Get all records from a given table following a condition.

**getRecordsAsDict**(*self*, *table*)

Acquires records using `getRecords()` and then leverages access using a dictionary data structure.

**newTable**(*self*, *tablename*, *\*args*)

**insertRecords**(*self*, *tablename*, *items*)

**insertRecord**(*self*, *tablename*, *record*)

**query**(*self*, *q*)



```
querymany(self, q, i)
```

```
close(self)
```

### *Inherited from object*

```
__delattr__(), __format__(), __getattr__(), __hash__(), __init__(),
__new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(),
__sizeof__(), __str__(), __subclasshook__()
```

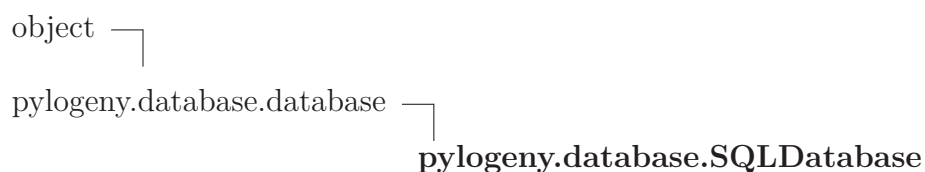
### 6.5.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 6.5.3 Class Variables

Name	Description
<code>cursor</code>	<b>Value:</b> None
<code>__abstractmethods__</code>	<b>Value:</b> frozenset(['close', 'getColumns', 'getTables', 'query', '...'])

## 6.6 Class *SQLDatabase*



Database object to allow reading from a MySQL database.

### 6.6.1 Methods

```
__init__(self, host, user, pw, db)
```

`x.__init__(...)` initializes `x`; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

<b>connect</b> ( <i>self</i> )
--------------------------------

<b>getTables</b> ( <i>self</i> )
----------------------------------

Overrides: <code>pylogeny.database.database.getTables</code>
--

<b>getColumns</b> ( <i>self</i> , <i>table</i> )
--

Return column information for a given table.
--

Overrides: <code>pylogeny.database.database.getColumns</code>
---

<b>query</b> ( <i>self</i> , <i>q</i> )
---

Overrides: <code>pylogeny.database.database.query</code>
--

<b>querymany</b> ( <i>self</i> , <i>q</i> , <i>i</i> )
--

Overrides: <code>pylogeny.database.database.querymany</code>
--

<b>close</b> ( <i>self</i> )
------------------------------

Overrides: <code>pylogeny.database.database.close</code>
--

### ***Inherited from `pylogeny.database.database` (Section 6.5)***

`filterRecords()`, `getHeaders()`, `getRecords()`, `getRecordsAsDict()`, `getRecordsColumn()`, `insertRecord()`, `insertRecords()`, `isEmpty()`, `iterRecords()`, `newTable()`

### ***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

#### **6.6.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

#### **6.6.3 Class Variables**

Name	Description
<code>__abstractmethods__</code>	<b>Value:</b> <code>frozenset([])</code>
<i>Inherited from <code>pylogeny.database.database</code> (Section 6.5)</i>	

*continued on next page*

Name	Description
cursor	

## 6.7 Class *SQLiteDatabase*



### 6.7.1 Methods

**\_\_init\_\_**(*self*, *filepath*)

x.**\_\_init\_\_**(...) initializes x; see help(type(x)) for signature

Overrides: object.**\_\_init\_\_** extit(inherited documentation)

**getColumns**(*self*, *table*)

Return column information for a given table.

Overrides: pylogeny.database.database.getColumns

**getTables**(*self*)

Overrides: pylogeny.database.database.getTables

**query**(*self*, *q*)

Overrides: pylogeny.database.database.query

**querymany**(*self*, *q*, *i*)

Overrides: pylogeny.database.database.querymany

**close**(*self*)

Overrides: pylogeny.database.database.close

**Inherited from *pylogeny.database.database* (Section 6.5)**

filterRecords(), getHeaders(), getRecords(), getRecordsAsDict(), getRecordsColumn(), insertRecord(), insertRecords(), isEmpty(), iterRecords(), newTable()

***Inherited from object***

```

__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),
__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),
__str__(), __subclasshook__()

```

**6.7.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**6.7.3 Class Variables**

Name	Description
<code>__abstractmethods__</code>	<b>Value:</b> <code>frozenset([])</code>
<i>Inherited from <code>pylogeny.database.database</code> (Section 6.5)</i>	
<code>cursor</code>	

## 7 Module *pylogeny.executable*

Defines an interface to manage interfacing with the system for respective application calls and implements some of these for executables such as FastTree and RAxML. Currently requires a UNIX-like environment (e.g., Mac OS X or a Linux-based environment).

### 7.1 Functions

<b>exeExists</b> ( <i>cmd</i> )
---------------------------------

Determines whether a function exists in a UNIX environment.
---

### 7.2 Variables

Name	Description
E_FASTTREE	<b>Value:</b> 'fasttree'
E_RAXML	<b>Value:</b> 'raxmlHPC'
E_TREPUZZ	<b>Value:</b> 'puzzle'
E_RSPP	<b>Value:</b> 'rspp'
L_TEMPDIR	<b>Value:</b> 'pylogeny'
__package__	<b>Value:</b> 'pylogeny'

### 7.3 Class *aTemporaryDirectory*

object └─ **pylogeny.executable.aTemporaryDirectory**

A class intended to be used as a context manager that allows Python to run in a temporary directory for a finite period of time.

#### 7.3.1 Methods

<b>__init__</b> ( <i>self</i> , <i>dir</i> =None)
---

x. <b>__init__</b> (...) initializes x; see help(type(x)) for signature
---

Overrides: object. <b>__init__</b> extit(inherited documentation)
---

<b>__enter__</b> ( <i>self</i> )
----------------------------------

<code>__exit__(self, *args)</code>
------------------------------------

**Inherited from object**

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

**7.3.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**7.4 Class executable**

**Known Subclasses:** `pylogeny.executable.consel`, `pylogeny.executable.fasttree`, `pylogeny.executable.raxml`, `pylogeny.executable.rspr`, `pylogeny.executable.treeppuzzle`

An abstract class for the instantiation and running of a single instance for a given application.

**7.4.1 Methods**

<code>getInstructionString(self)</code>
---

<code>run(self)</code>
------------------------

Perform a run of this application.
------------------------------------

**Inherited from object**

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__init__()`,  
`__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`,  
`__sizeof__()`, `__str__()`, `__subclasshook__()`

**7.4.2 Properties**

Name	Description
<i>Inherited from object</i> __class__	

### 7.4.3 Class Variables

Name	Description
exeName	<b>Value:</b> None
__abstractmethods__	<b>Value:</b> frozenset(['getInstructionString'])

## 7.5 Class treepuzzle

object └─

pylogeny.executable.executable └─  
pylogeny.executable.treepuzzle

Wrap TREE-PUZZLE in order to create an intermediate file for CONSEL to read and assign confidence to a set of trees. Requires TREE-PUZZLE to be installed.

### 7.5.1 Methods

<b>__init__</b> ( <i>self</i> , <i>ali</i> , <i>treefile</i> ) x.__init__(...) initializes x; see help(type(x)) for signature Overrides: object.__init__ extit(inherited documentation)
<b>getInstructionString</b> ( <i>self</i> ) Overrides: pylogeny.executable.executable.getInstructionString
<b>getSiteLikelihoodFile</b> ( <i>self</i> )

*Inherited from pylogeny.executable.executable(Section 7.4)*

run()

*Inherited from object*

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(),

`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

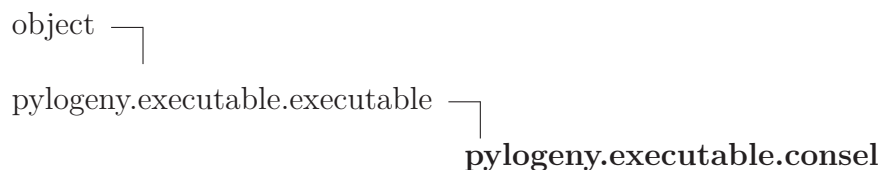
### 7.5.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 7.5.3 Class Variables

Name	Description
<code>exeName</code>	<b>Value:</b> 'puzzle'
<code>__abstractmethods__</code>	<b>Value:</b> frozenset([])

## 7.6 Class *consel*



Denotes a single run of the CONSEL workflow in order to acquire a confidence interval and perform an AU test on a set of trees. Requires CONSEL to be installed.

### 7.6.1 Methods

<b><code>__init__</code></b> ( <i>self</i> , <i>treeset</i> , <i>alignment</i> , <i>name</i> )
x. <b><code>__init__</code></b> (...) initializes x; see <code>help(type(x))</code> for signature
Overrides: <code>object.__init__</code> <code>extit</code> (inherited documentation)
<b><code>getInstructionString</code></b> ( <i>self</i> )
Overrides: <code>pylogeny.executable.executable.getInstructionString</code>
<b><code>getInterval</code></b> ( <i>self</i> )
Compute the AU test. Return the interval of trees.

*Inherited from `pylogeny.executable.executable` (Section 7.4)*



`run()`

### *Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

### 7.6.2 Properties

Name	Description
<i>Inherited from object</i> <code>__class__</code>	

### 7.6.3 Class Variables

Name	Description
<code>__abstractmethods__</code>	<b>Value:</b> <code>frozenset([])</code>
<i>Inherited from <code>pylogeny.executable.executable</code> (Section 7.4)</i> <code>exeName</code>	

## 7.7 Class *fasttree*

object └

pylogeny.executable.executable └  
**pylogeny.executable.fasttree**

Denotes a single run of the FastTree executable in order to acquire an approximate maximum likelihood tree for the input alignment. See <http://www.microbesonline.org/fasttree/> for more information on FastTree. Requires FastTree to be installed.

### 7.7.1 Methods

`__init__(self, inp_align, out_file=None, isProtein=True)`  
`x.__init__(...)` initializes x; see `help(type(x))` for signature  
 Overrides: `object.__init__` `exitit`(inherited documentation)

**getInstructionString**(*self*)

Overrides: pylogeny.executable.executable.getInstructionString

*Inherited from pylogeny.executable.executable(Section 7.4)*

run()

*Inherited from object*

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(),  
 \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_(),  
 \_\_str\_\_(), \_\_subclasshook\_\_()

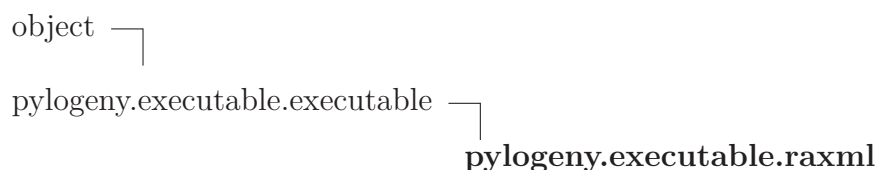
### 7.7.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

### 7.7.3 Class Variables

Name	Description
exeName	<b>Value:</b> 'fasttree'
__abstractmethods__	<b>Value:</b> frozenset([])

## 7.8 Class raxml



Denotes a single run of the RAxML executable. See <http://sco.h-its.org/exelixis/software.html> for more information on RAxML. Requires RAxML to be installed.

### 7.8.1 Methods

```
__init__(self, inp_align, out_file, model=None, is_Protein=True,
interTrees=False, alg=None, startingTree=None, rapid=False, slow=False,
optimizeBootstrap=False, numboot=100, log=None, wdir=None)
```

x.\_\_init\_\_(...) initializes x; see help(type(x)) for signature

Overrides: object.\_\_init\_\_ extit(inherited documentation)

```
getInstructionString(self)
```

Overrides: pylogeny.executable.executable.getInstructionString

```
runFunction(self, alg)
```

*Inherited from pylogeny.executable.executable(Section 7.4)*

```
run()
```

*Inherited from object*

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),
__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),
__str__(), __subclasshook__()
```

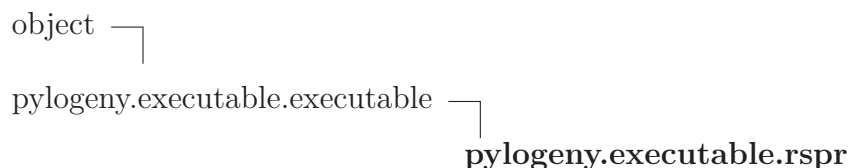
### 7.8.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

### 7.8.3 Class Variables

Name	Description
exeName	<b>Value:</b> 'raxmlHPC'
__abstractmethods__	<b>Value:</b> frozenset([])

## 7.9 Class *rspr*



Denotes a single run of the rSPR executable by Dr. Chris Whidden (2014), a software package for computing rooted subtree-prune-and-regraft (SPR) distances. See <http://kiwi.cs.dal.ca/Software/RSPR>. Requires the executable to be on PATH.

### 7.9.1 Methods

<b><code>__init__(self, treeA, treeB, algorithm='', overlap=True)</code></b>
Algorithm choices are defined in this class. If overlap is set to True, will attempt to consolidate taxa names such that they are overlapping (otherwise, RSPR will return an error if they do not match). Overrides: <code>object.__init__</code>
<b><code>getSPRDistance(self)</code></b>
<b><code>getInstructionString(self)</code></b>
Overrides: <code>pylogeny.executable.executable.getInstructionString</code>

*Inherited from `pylogeny.executable.executable` (Section 7.4)*

`run()`

*Inherited from `object`*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

### 7.9.2 Properties

Name	Description
<i>Inherited from <code>object</code></i>	
<code>__class__</code>	

### 7.9.3 Class Variables

Name	Description
exeName	<b>Value:</b> 'rspr'
RSPR_ALG_DEFAULT	<b>Value:</b> ''
RSPR_ALG_APPROX	<b>Value:</b> '-approx'
RSPR_ALG_BB	<b>Value:</b> '-bb'
RSPR_ALG_FPT	<b>Value:</b> '-fpt'
__abstractmethods__	<b>Value:</b> frozenset([])

## 8 Module `pylogeny.heuristic`

Define the interface for a heuristic in order to implement any manner of heuristic for a combinatorial problem that can be abstracted into a state graph. In this case, a phylogenetic tree space.

### 8.1 Variables

Name	Description
SMGR_MIN_NUM_LIKELIHOOD	<b>Value:</b> 32
__package__	<b>Value:</b> 'pylogeny'

### 8.2 Class `heuristic`

object —  
     **`pylogeny.heuristic.heuristic`**

**Known Subclasses:** `pylogeny.heuristic.phylogeneticLinearHeuristic`

A base interface for a heuristic that explores a state graph.

#### 8.2.1 Methods

**`__init__(self, G=None, start=None)`**

x.`__init__`(...) initializes x; see `help(type(x))` for signature

Overrides: object.`__init__` `__exit__`(inherited documentation)

**`explore(self)`**

**`getStateGraph(self)`**

**`getStartState(self)`**

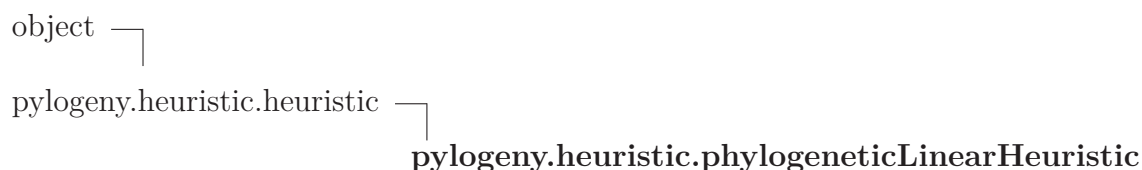
*Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

### 8.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

## 8.3 Class *phylogeneticLinearHeuristic*



**Known Subclasses:** *pylogeny.heuristic.RAxMLIdentify*, *pylogeny.heuristic.likelihoodGreedy*, *pylogeny.heuristic.parsimonyGreedy*, *pylogeny.heuristic.smoothGreedy*

A base class for a heuristic that works on a phylogenetic landscape and only possesses a single path (of search).

### 8.3.1 Methods

`__init__(self, ls, startNode)`

`x.__init__(...)` initializes x; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

`getPath(self)`

`getBestTree(self)`

*Inherited from `pylogeny.heuristic.heuristic` (Section 8.2)*

`explore()`, `getStartState()`, `getStateGraph()`

*Inherited from `object`*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

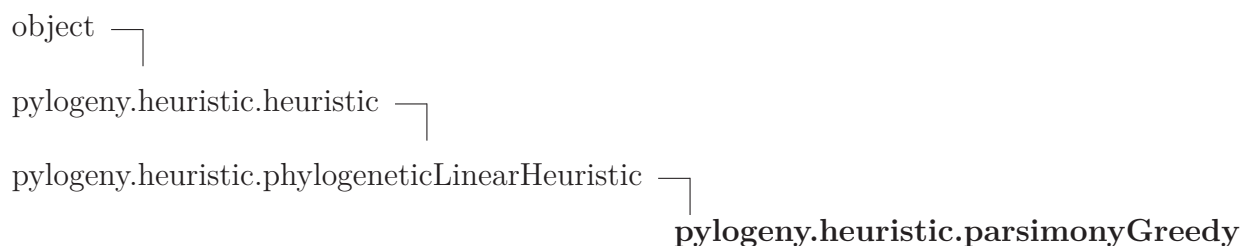
### 8.3.2 Properties

Name	Description
<i>Inherited from object</i> __class__	

### 8.3.3 Class Variables

Name	Description
bestTree	<b>Value:</b> None
path	<b>Value:</b> []

## 8.4 Class *parsimonyGreedy*



Greedy (hill-climbing) landscape exploration by comparsion of parsimony.

### 8.4.1 Methods

<b>__init__</b> ( <i>self</i> , <i>ls</i> , <i>startNode</i> )
x. <b>__init__</b> (...) initializes x; see help(type(x)) for signature Overrides: object. <b>__init__</b> extit(inherited documentation)
<b>explore</b> ( <i>self</i> )
Perform greedy search of the landscape using a method of greed via parsimonious criterion. Overrides: pylogeny.heuristic.heuristic.explore

*Inherited from pylogeny.heuristic.phylogeneticLinearHeuristic(Section 8.3)*

getBestTree(), getPath()

*Inherited from pylogeny.heuristic.heuristic(Section 8.2)*

getStartState(), getStateGraph()



***Inherited from object***

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),
__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),
__str__(), __subclasshook__()
```

**8.4.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**8.4.3 Class Variables**

Name	Description
<i>Inherited from <code>pylogeny.heuristic.phylogeneticLinearHeuristic</code> (Section 8.3)</i>	
<code>bestTree</code> , <code>path</code>	

**8.5 Class `likelihoodGreedy`**

```

object └─
pylogeny.heuristic.heuristic └─
pylogeny.heuristic.phylogeneticLinearHeuristic └─
                                                    pylogeny.heuristic.likelihoodGreedy
```

Greedy (hill-climbing) landscape exploration by comparsion of likelihood.

**8.5.1 Methods**

<b><code>__init__(self, ls, startNode)</code></b>
<code>x.__init__(...)</code> initializes x; see <code>help(type(x))</code> for signature
Overrides: <code>object.__init__</code> <code>exitit</code> (inherited documentation)

<b><code>explore(self)</code></b>
Perform greedy search of the landscape using a method of greed via likelihood.
Overrides: <code>pylogeny.heuristic.heuristic.explore</code>

*Inherited from pylogeny.heuristic.phylogeneticLinearHeuristic (Section 8.3)*

getBestTree(), getPath()

*Inherited from pylogeny.heuristic.heuristic (Section 8.2)*

getStartState(), getStateGraph()

*Inherited from object*

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(),  
\_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_(),  
\_\_str\_\_(), \_\_subclasshook\_\_()

### 8.5.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

### 8.5.3 Class Variables

Name	Description
<i>Inherited from pylogeny.heuristic.phylogeneticLinearHeuristic (Section 8.3)</i>	
bestTree, path	

## 8.6 Class smoothGreedy

```

object └─
pylogeny.heuristic.heuristic └─
pylogeny.heuristic.phylogeneticLinearHeuristic └─
                                                    pylogeny.heuristic.smoothGreedy

```

Parsimony-driven greedy landscape exploration by comparsion of likelihoods.

### 8.6.1 Methods

**\_\_init\_\_**(*self*, *ls*, *startNode*)

*x*.**\_\_init\_\_**(...) initializes *x*; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

**explore**(*self*)

Perform greedy search of the landscape using a method of greed via parsimonious criterion and then performing final smoothing via likelihood on top 10% of 1-SPR neighbors ranked on basis of parsimony.

Overrides: `pylogeny.heuristic.heuristic.explore`

*Inherited from `pylogeny.heuristic.phylogeneticLinearHeuristic` (Section 8.3)*

`getBestTree()`, `getPath()`

*Inherited from `pylogeny.heuristic.heuristic` (Section 8.2)*

`getStartState()`, `getStateGraph()`

*Inherited from `object`*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

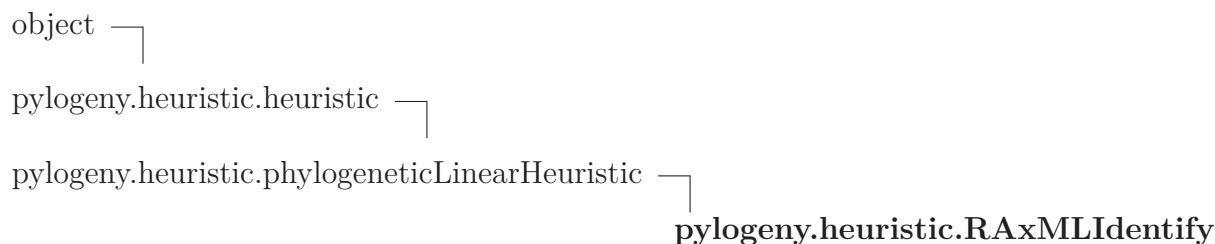
### 8.6.2 Properties

Name	Description
<i>Inherited from <code>object</code></i>	
<code>__class__</code>	

### 8.6.3 Class Variables

Name	Description
<i>Inherited from <code>pylogeny.heuristic.phylogeneticLinearHeuristic</code> (Section 8.3)</i>	
<code>bestTree</code> , <code>path</code>	

## 8.7 Class RAxMLIdentify



RAxML-driven landscape evaluation of intermediate checkpoint trees output from the RAxML executable.

### 8.7.1 Methods

```
__init__(self, ls, startNode, workdir='.raxml')
```

x.**\_\_init\_\_**(...) initializes x; see help(type(x)) for signature

Overrides: object.**\_\_init\_\_** extit(inherited documentation)

```
__setupWorkDir__(self)
```

```
__setupExecutable__(self)
```

```
__readLogFile__(self)
```

```
__readIterTrees__(self, iters)
```

```
explore(self)
```

Overrides: pylogeny.heuristic.heuristic.explore

**Inherited from pylogeny.heuristic.phylogeneticLinearHeuristic (Section 8.3)**

getBestTree(), getPath()

**Inherited from pylogeny.heuristic.heuristic (Section 8.2)**

getStartState(), getStateGraph()

**Inherited from object**

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),
__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),
__str__(), __subclasshook__()
```

### 8.7.2 Properties

Name	Description
<i>Inherited from object</i> __class__	

### 8.7.3 Class Variables

Name	Description
<i>Inherited from pylogeny.heuristic.phylogeneticLinearHeuristic (Section 8.3)</i> bestTree, path	

## 9 Module *pylogeny.landscape*

Encapsulate a phylogenetic tree space. A phylogenetic landscape or tree space refers to the entire combinatorial space comprising all possible phylogenetic tree topologies for a set of  $n$  taxa. The landscape of  $n$  taxa can be defined as consisting of a finite set  $T$  of tree topologies. Tree topologies can be associated with a fitness function  $f(t_i)$  describing their fit. This forms a discrete solution search space and finite graph  $(T, E) = G$ .  $E(G)$  refers to the neighborhood relation on  $T(G)$ . Edges in this graph are bidirectional and represent transformation from one tree topology to another by a tree rearrangement operator. An edge between  $t_i$  and  $t_j$  would be notated as  $e_{\{ij\}}$  in  $E(G)$ .

### 9.1 Variables

Name	Description
LS_NOT_DEFINED	<b>Value:</b> -1
__package__	<b>Value:</b> 'pylogeny'

### 9.2 Class graph

object —  
**pylogeny.landscape.graph**

**Known Subclasses:** *pylogeny.landscape.landscape*

Define an empty graph object.

#### 9.2.1 Methods

<b>__init__</b> ( <i>self</i> , <i>gr</i> =None)
Instantiate a graph.
:param <i>gr</i> : A networkx graph object, if already exists.
Overrides: object.__init__
<b>getNetworkXObject</b> ( <i>self</i> )
Return the internal networkx graph object.
<b>__len__</b> ( <i>self</i> )

<b>__iter__</b> ( <i>self</i> )
<b>getSize</b> ( <i>self</i> ) Return the number of nodes in the graph.
<b>getNodeNames</b> ( <i>self</i> ) Return the names of nodes in the graph.
<b>iterNodes</b> ( <i>self</i> ) Iterate over all node keys.
<b>getNodes</b> ( <i>self</i> )
<b>getEdges</b> ( <i>self</i> )
<b>getEdgesFor</b> ( <i>self</i> , <i>i</i> )
<b>getNode</b> ( <i>self</i> , <i>i</i> )
<b>getEdge</b> ( <i>self</i> , <i>i</i> , <i>j</i> )
<b>isEdge</b> ( <i>self</i> , <i>i</i> , <i>j</i> )
<b>getNeighborsFor</b> ( <i>self</i> , <i>i</i> )
<b>getDegreeFor</b> ( <i>self</i> , <i>i</i> ) Return in- and out-degree for node named <i>i</i> .
<b>setDefaultWeight</b> ( <i>self</i> , <i>w</i> )
<b>clearEdgeWeights</b> ( <i>self</i> )
<b>getNumComponents</b> ( <i>self</i> ) Get the number of components of the graph.
<b>getComponents</b> ( <i>self</i> ) Get the connected components in the graph.

<b>getComponentOfNode</b> ( <i>self</i> , <i>i</i> )
--

Get the graph component of a given node.
--

<b>getCliques</b> ( <i>self</i> )
-----------------------------------

Get the cliques present in the graph.
---------------------------------------

<b>getCliqueNumber</b> ( <i>self</i> )
--

Get the clique number of the graph.
-------------------------------------

<b>getNumCliques</b> ( <i>self</i> )
--------------------------------------

Get the number of cliques found in the graph.
---

<b>getCliquesOfNode</b> ( <i>self</i> , <i>i</i> )
--

Get the clique that a node corresponds to.
--

<b>getCenter</b> ( <i>self</i> )
----------------------------------

Get the centre of the graph.
------------------------------

<b>getDiameter</b> ( <i>self</i> )
------------------------------------

Acquire the diameter of the graph.
------------------------------------

<b>getMST</b> ( <i>self</i> )
-------------------------------

Acquire the minimum spanning tree for the graph.
--

<b>hasPath</b> ( <i>self</i> , <i>nodA</i> , <i>nodB</i> )
--

See if a path exists between two nodes.
---

<b>getShortestPath</b> ( <i>self</i> , <i>nodA</i> , <i>nodB</i> )
--

Get the shortest path between two nodes.
--

<b>getShortestPathLength</b> ( <i>self</i> , <i>nodA</i> , <i>nodB</i> )
--

Get the shortest path length between two nodes.
---

### *Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,

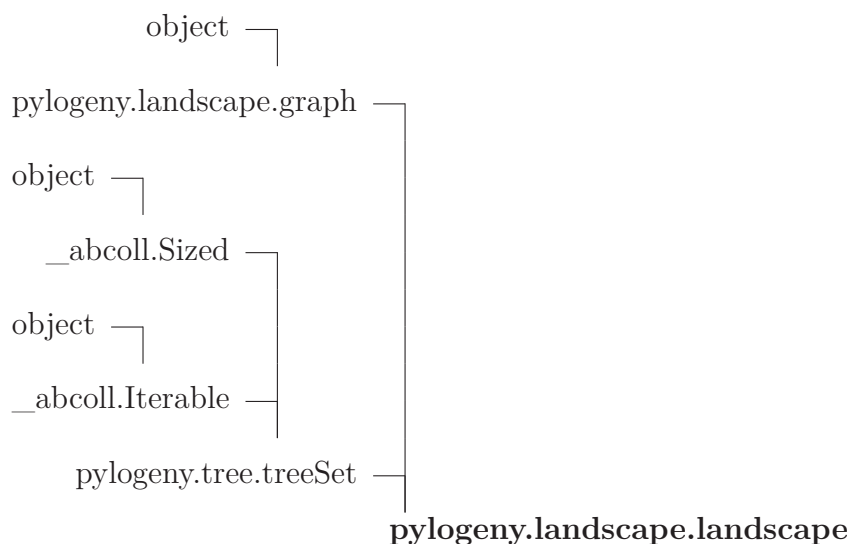


`__str__()`, `__subclasshook__()`

### 9.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

## 9.3 Class landscape



**Known Subclasses:** `pylogeny.database.DatabaseLandscape`, `pylogeny.database.SQLiteLandscape`  
 Defines an entire phylogenetic tree space.

### 9.3.1 Methods

<b><code>__init__</code></b> ( <i>self</i> , <i>ali</i> , <i>starting_tree</i> =None, <i>root</i> =True, <i>operator</i> ='SPR')
Initialize the landscape.
:param <i>ali</i> : An :class:'alignment.alignment' object. :param <i>starting_tree</i> : An optional tree object to start the landscape with. :param <i>root</i> : Whether or not to acquire an approximate maximum likelihood tree (FastTree) or start the landscape with a given starting tree. :param <i>operator</i> : A string that describes what operator the landscape is mostly comprised of.
Overrides: <code>object.__init__</code>

**getAlignment**(*self*)

Acquire the alignment object associated with this space.

**getNumberTaxa**(*self*)

Return the number of different taxa present in any respective tree in the landscape.

**getPossibleNumberRootedTrees**(*self*)

Assuming all of the trees in the space are rooted, return the maximum possible number of unrooted trees that can possibly be generated for the number of taxa of trees in the landscape.

**getPossibleNumberUnrootedTrees**(*self*)

Assuming all of the trees in the space are unrooted, return the maximum possible number of unrooted trees that can possibly be generated for the number of taxa of trees in the landscape.

**getRoot**(*self*)

Returns the index to the root (starting) tree of the space.

**getRootTree**(*self*)

Acquire the first tree that was placed in this space.

**setAlignment**(*self*, *ali*)

Set the alignment present in this landscape. WARNING; will not modify existing scores.

**setOperator**(*self*, *op*)

Set the operator assigned to this landscape.

**getTree**(*self*, *i*)

Get the object for a tree by its name.

**\_\_iter\_\_**(*self*)

Overrides: `__abcoll.Iterable.__iter__`

---

**getVertex**(*self*, *i*)

---

Acquire a vertex object from the landscape; this is a high-level representation of a tree in the landscape with additional functionality. Object created upon invocation of this function.

---

**removeTreeByIndex**(*self*, *i*)

---

Remove a tree from the landscape by index.

---

**removeTree**(*self*, *tree*)

---

Remove a tree from the landscape by object.

Overrides: `pylogeny.tree.treeSet.removeTree`

---

**addTreeByNewick**(*self*, *newick*, *score=True*)

---

Add a tree to the landscape by Newick string. Will return an index.

Overrides: `pylogeny.tree.treeSet.addTreeByNewick`

---

**addTree**(*self*, *tr*, *score=True*)

---

Add a tree to the landscape. Will return its index.

Overrides: `pylogeny.tree.treeSet.addTree`

---

**exploreRandomTree**(*self*, *i*, *type=1*)

---

Acquire a single neighbor to a tree in the landscape by performing a random rearrangement of type SPR (by default), NNI, or TBR – this is done by performing a rearrangement on a random branch in the topology.

Rearrangement type is provided as a rearrangement module type definition of form, for example, `TYPE_SPR`, `TYPE_NNI`, etc.

---

**exploreTree**(*self*, *i*, *type=1*)

---

Get all neighbors to a tree named *i* in the landscape using a respective rearrangement operator as defined in the rearrangement module.

Rearrangement type is provided as a rearrangement module type definition of form, for example, `TYPE_SPR`, `TYPE_NNI`, etc. By default, this is `TYPE_SPR`.

---

**getLocks**(*self*)

---

**toggleLock**(*self*, *lock*)

Add a bipartition to the list of locked bipartitions if not present; otherwise, remove it. Return status of lock.

**lockBranchFoundInTree**(*self*, *tr*, *br*)

Given a tree node and a branch object, add a given bipartition to the bipartition lock list. Returns true if locked.

**getBipartitionFoundInTreeByIndex**(*self*, *tr*, *brind*, *topol=None*)

Given a tree node and a branch index, return the associated bipartition.

**lockBranchFoundInTreeByIndex**(*self*, *tr*, *brind*)

Given a tree node and a branch index, add a given bipartition to the bipartition lock list. Returns true if locked.

**isViolating**(*self*, *i*)

Determine if a tree is violating any locks intrinsic to the landscape.

**\_\_getitem\_\_**(*self*, *i*)

Overrides: `pylogeny.tree.treeSet.__getitem__`

**indexOf**(*self*, *tr*)

Acquire the index/name in this landscape of a tree object. Returns -1 if not found.

Overrides: `pylogeny.tree.treeSet.indexOf`

**findTree**(*self*, *newick*)

Find a tree by Newick string, taking into account branch lengths. Returns the name of this tree in the landscape.

**findTreeTopology**(*self*, *newick*)

Find a tree by topology, not taking into account branch lengths.

**findTreeTopologyByStructure**(*self*, *struct*)

Find a tree by topology, not taking into account branch lengths, given the topology.

**getBestImprovement**(*self*, *i*)

For a tree in the landscape, investigate neighbors to find a tree that leads to the best improvement of fitness function score on the basis of likelihood.

**getPathOfBestImprovement**(*self*, *i*)

For a tree in the landscape, investigate neighbors iteratively until a best path of score improvement is found on basis of likelihood.

**getAllPathsOfBestImprovement**(*self*)

Return all paths of best improvement as a dictionary.

**iterAllPathsOfBestImprovement**(*self*)

Return an iterator for all paths of best improvement.

**isLocalOptimum**(*self*, *i*)

Determine if a tree is, without any doubt, a local optimum.

**getLocalOptima**(*self*)

Get all trees in the landscape that can be labelled as a local optimum.

**getGlobalOptimum**(*self*)

Get the global optimum of the current space.

**\_\_str\_\_**(*self*)

str(x)

Overrides: object.\_\_str\_\_ extit(inherited documentation)

**toProperNewickTreeSet**(*self*)

Convert this landscape into an unorganized set of trees where taxa names are transformed to their original form ( i.e. not transformed to a state friendly for the Phylip format).

**toTreeSet**(*self*)

Convert this landscape into an unorganized set of trees.

*Inherited from pylogeny.landscape.graph(Section 9.2)*

`__len__()`, `clearEdgeWeights()`, `getCenter()`, `getCliqueNumber()`, `getCliques()`, `getCliquesOfNode()`, `getComponentOfNode()`, `getComponents()`, `getDegreeFor()`, `getDiameter()`, `getEdge()`, `getEdges()`, `getEdgesFor()`, `getMST()`, `getNeighborsFor()`, `getNetworkXObject()`, `getNode()`, `getNodeNames()`, `getNodes()`, `getNumCliques()`, `getNumComponents()`, `getShortestPath()`, `getShortestPathLength()`, `getSize()`, `hasPath()`, `isEdge()`, `iterNodes()`, `setDefaultWeight()`

***Inherited from `pylogeny.tree.treeSet` (Section 17.4)***

`__setitem__()`, `fromTreeFile()`, `iterTrees()`, `toTreeFile()`

***Inherited from `__abcoll.Sized`***

`__subclasshook__()`

***Inherited from object***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`

### 9.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 9.3.3 Class Variables

Name	Description
<i>Inherited from <code>pylogeny.tree.treeSet</code> (Section 17.4)</i>	
<code>__abstractmethods__</code>	

## 9.4 Class vertex

```

object └─
        pylogeny.landscape.vertex

```

Encapsulate a single vertex in the landscape and add convenient functionality to alias parent landscape functions.

### 9.4.1 Methods

**\_\_init\_\_**(*self*, *obj*, *ls*)

*x*.**\_\_init\_\_**(...) initializes *x*; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)

**getIndex**(*self*)

**getDict**(*self*)

**getObject**(*self*)

**getTree**(*self*)

**getNewick**(*self*)

**getScore**(*self*)

**getOrigin**(*self*)

**getNeighbors**(*self*)

**getDegree**(*self*)

**isLocalOptimum**(*self*)

**isExplored**(*self*)

**isFailed**(*self*)

**setExplored**(*self*, *exp*)

Sets the "explored" flag of this node in the landscape.

**approximatePossibleNumNeighbors**(*self*)

Approximate the possible number of neighbors to this vertex by considering the type of tree rearrangement operator.

**scoreLikelihood**(*self*)

Acquire the log-likelihood for this vertex.

**getBestImprovement**(*self*)

Alias function for function of same name in parent landscape.

**getPathOfBestImprovement**(*self*)

Alias function for function of same name in parent landscape.

**isBestImprovement**(*self*)

Check to see if this vertex is a best move for another node.

**isViolating**(*self*)

Alias function for function of same name in parent landscape.

**getProperNewick**(*self*)

Get the proper Newick string for a tree. :returns: A string.

**iterBipartitions**(*self*)

Return a generator to iterate over all bipartitions for this vertex.

**getBipartitions**(*self*)

Get all bipartitions for this vertex.

**getBipartitionScores**(*self*)

Get all corresponding bipartition vectors of SPR scores.

**getNeighborsOfBipartition**(*self*, *bi*)

Get corresponding neighbors of a bipartition in this vertex's tree.

**getNeighborsOfBranch**(*self*, *br*)

Get corresponding neighbors of a branch in this vertex's tree.

***Inherited from object***

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(),  
 \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_(),  
 \_\_str\_\_(), \_\_subclasshook\_\_()



### 9.4.2 Properties

Name	Description
<i>Inherited from object</i> <code>__class__</code>	

## 10 Module `pylogeny.landscapeWriter`

Serialize a phylogenetic landscape into an SQLite database file made up of three components: all tree IDs and respective scores, the alignment file as a set of sequences, and a representation of the graph as an edge list.

### 10.1 Variables

Name	Description
<code>__package__</code>	Value: <code>'pylogeny'</code>

### 10.2 Class `landscapeWriter`

object —  
     **`pylogeny.landscapeWriter.landscapeWriter`**

**Known Subclasses:** `pylogeny.JSONWriter.JSONWriter`

Encapsulate the writing of a landscape to a file format.

#### 10.2.1 Methods

<b><code>__init__(self, landscape, name)</code></b> <code>x.__init__(...)</code> initializes <code>x</code> ; see <code>help(type(x))</code> for signature Overrides: <code>object.__init__</code> <code>__init__</code> (inherited documentation)
<b><code>writeFile(self, path='.')</code></b> Write the landscape serialized file to given path.

**Inherited from `object`**

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

#### 10.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

### 10.3 Class `landscapeParser`

object —  
**`pylogeny.landscapeWriter.landscapeParser`**

Encapsulates the construction of a landscape object from a sqlite landscape file.

#### 10.3.1 Methods

<b><code>__init__(self, path)</code></b>
<code>x.__init__(...)</code> initializes x; see <code>help(type(x))</code> for signature
Overrides: <code>object.__init__</code> <code>exitit</code> (inherited documentation)

<b><code>getName(self)</code></b>
Acquire the name of the parsed landscape.

<b><code>parse(self)</code></b>
Parse the file.

#### *Inherited from object*

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__str__()`, `__subclasshook__()`

#### 10.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

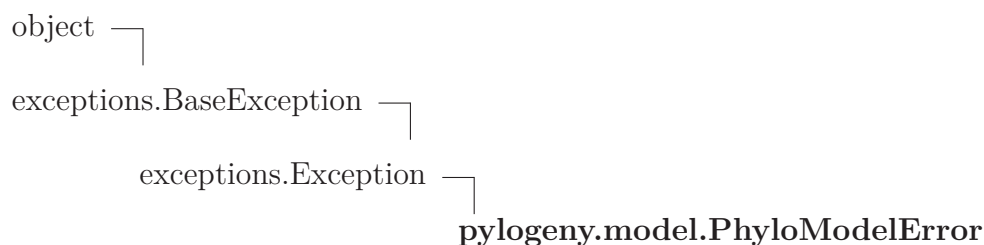
## 11 Module *pylogeny.model*

Phylogenetic tree scoring models; intended to be coupled with the use of *pytbeaglehon* (BEAGLE) high-performance library.

### 11.1 Variables

Name	Description
<code>pytbeaglehonEnabled</code>	<b>Value:</b> <code>False</code>
<code>__package__</code>	<b>Value:</b> <code>'pylogeny'</code>

### 11.2 Class *PhyloModelError*



#### 11.2.1 Methods

**`__init__`**(*self*, *v*)

*x*.**`__init__`**(...) initializes *x*; see `help(type(x))` for signature

Overrides: `object.__init__` `extit`(inherited documentation)

**`__str__`**(*self*)

`str(x)`

Overrides: `object.__str__` `extit`(inherited documentation)

*Inherited from `exceptions.Exception`*

**`__new__`**()

*Inherited from `exceptions.BaseException`*

**`__delattr__`**(), **`__getattr__`**(), **`__getitem__`**(), **`__getslice__`**(), **`__reduce__`**(), **`__repr__`**(), **`__setattr__`**(), **`__setstate__`**(), **`__unicode__`**()

***Inherited from object***

`__format__()`, `__hash__()`, `__reduce_ex__()`, `__sizeof__()`, `__subclasshook__()`

**11.2.2 Properties**

Name	Description
<i>Inherited from exceptions.BaseException</i>	
<code>args</code> , <code>message</code>	
<i>Inherited from object</i>	
<code>__class__</code>	

**11.3 Class *DiscreteStateModel***

object —  
**pylogeny.model.DiscreteStateModel**

Initialize a discrete state model for phylogenetic data. State frequencies and character time are determined from the given alignment object.

**11.3.1 Methods**

<code>__init__(self, alignment)</code>
x. <code>__init__</code> (...) initializes x; see <code>help(type(x))</code> for signature Overrides: <code>object.__init__</code> <code>exitit</code> (inherited documentation)
<code>getAlignment(self)</code>
<code>getAlignmentAsStateList(self)</code>
<code>getSequenceMatrix(self)</code>
<code>getCharType(self)</code>
<code>getStateFreqs(self)</code>
<code>getRawStateFreqs(self)</code>

<code>getRawStateFreqsAsList(<i>self</i>)</code>
--

<code>getRawStateFreqsAsDict(<i>self</i>)</code>
--

<code>getFrequencyOfState(<i>self</i>, <i>i</i>)</code>
---

<code>getRawFrequencyOfState(<i>self</i>, <i>i</i>)</code>
--

***Inherited from object***

```
__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),
__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),
__str__(), __subclasshook__()
```

**11.3.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

## 12 Module *pylogeny.newick*

Newick string parsing and object interaction. A Newick string can represent a phylogenetic tree.

### 12.1 Functions

#### **assignParents**(*top*)

Should be a one-time use function. Goes through and assigns parents to the parsed newick tree structure nodes and branches to allow for up-traversal.

#### **removeBranchLengths**(*top*)

Goes through and removes any stored branch lengths.

#### **removeUnaryInternalNodes**(*top*)

Goes through and ensures any degree-2 internal nodes are smoothed into a single degree-3 internal node.

#### **invertAlongPathToNode**(*target, top*)

DANGEROUS: Reverses all directionality to a given node from a top-level node. Intended as a low-level function for rerooting a tree.

#### **shuffleLeaves**(*top*)

DANGEROUS: Given a top-level node, shuffle all leaves in this tree.

#### **getAllBranches**(*br*)

Given a branch, traverse subtree and return comprising branches as a list.

#### **isSibling**(*br, other*)

Given a branch, determine if that branch is adjacent to another branch.

#### **getBalancingBracket**(*newick, i*)

Given a position of an opening bracket in a newick string, *i*, output the closing bracket's position that corresponds to this opening bracket.

**getBranchLength**(*newick*, *i*)

Given a position of a colon symbol (indicating a branch length), return the branch length.

**getLeafName**(*newick*, *i*)

Given the position of a leaf, find its complete name.

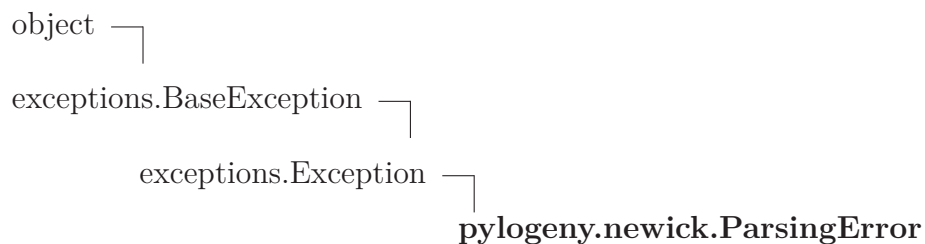
**parseNewick**(*newick*, *i*, *j*, *top*)

Parse a newick string into a topological newick structure given a top-level node.

## 12.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> 'pylogeny'

## 12.3 Class *ParsingError*



### 12.3.1 Methods

**\_\_init\_\_**(*self*, *val*)

*x*.**\_\_init\_\_**(...) initializes *x*; see `help(type(x))` for signature

Overrides: `object.__init__` `extit`(inherited documentation)

**\_\_str\_\_**(*self*)

`str(x)`

Overrides: `object.__str__` `extit`(inherited documentation)

*Inherited from exceptions.Exception*



`__new__()`

*Inherited from exceptions.BaseException*

`__delattr__()`, `__getattr__()`, `__getitem__()`, `__getslice__()`, `__reduce__()`, `__repr__()`, `__setattr__()`, `__setstate__()`, `__unicode__()`

*Inherited from object*

`__format__()`, `__hash__()`, `__reduce_ex__()`, `__sizeof__()`, `__subclasshook__()`

### 12.3.2 Properties

Name	Description
<i>Inherited from exceptions.BaseException</i>	
args, message	
<i>Inherited from object</i>	
<code>__class__</code>	

## 12.4 Class node

```

object └─
pylogeny.base.treeNode └─
                        pylogeny.newick.node

```

Newick node.

### 12.4.1 Methods

`__init__(self, lbl='', children=None, parent=None)`  
 x.`__init__`(...) initializes x; see help(type(x)) for signature  
 Overrides: object.`__init__` extit(inherited documentation)

`__str__(self)`  
 str(x)  
 Overrides: object.`__str__` extit(inherited documentation)

*Inherited from pylogeny.base.treeNode(Section 5.4)*

addChild(), getChildByIndex(), getChildren(), getLabel(), getParent(), isInternalNode(), isLeaf()

### *Inherited from object*

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(),  
 \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_(),  
 \_\_subclasshook\_\_()

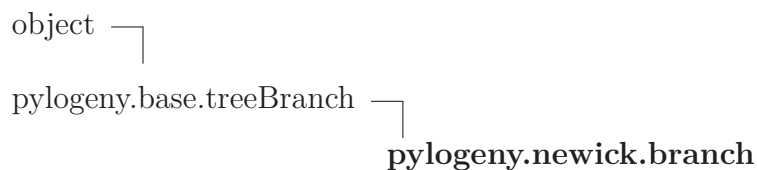
#### 12.4.2 Properties

Name	Description
<i>Inherited from object</i> __class__	

#### 12.4.3 Class Variables

Name	Description
<i>Inherited from pylogeny.base.treeNode (Section 5.4)</i> children, label, parent	

## 12.5 Class branch



Newick branch.

#### 12.5.1 Methods

<b>__init__</b> ( <i>self, chi, l, parent=None, s=None</i> )
x. <b>__init__</b> (...) initializes x; see help(type(x)) for signature
Overrides: object. <b>__init__</b> extit(inherited documentation)

<code>__str__(self)</code>  <code>str(x)</code>  Overrides: <code>object.__str__</code> <code>exitit</code> (inherited documentation)
---

***Inherited from `pylogeny.base.treeBranch`(Section 5.5)***

`getChild()`, `getLabel()`, `getParent()`

***Inherited from `object`***

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`,  
`__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`,  
`__subclasshook__()`

### 12.5.2 Properties

Name	Description
<i>Inherited from <code>object</code></i>	
<code>__class__</code>	

### 12.5.3 Class Variables

Name	Description
<i>Inherited from <code>pylogeny.base.treeBranch</code> (Section 5.5)</i>	
<code>child</code> , <code>label</code> , <code>parent</code>	

## 12.6 Class `newickParser`

Parsing object for Newick strings.

### 12.6.1 Methods

<code>__init__(self, newick)</code>
-------------------------------------

<b><code>parse(self)</code></b>
Parse the stored newick string into a topological structure.

<code>__str__(self)</code>
----------------------------

## 13 Module `pylogeny.parsimony`

Toolkit for performance of Parsimonious Criterion (Parsimony) methods of optimization of a phylogenetic topology with a particular set of data.

### 13.1 Functions

**`fitch_cost`**(*topology*, *profiles*)

Calculate the cost using Fitch algorithm on profile set and alignment.  
 Deprecated: Python implementation of the Fitch algorithm; see `fitch C++` module for a C++ implementation that is roughly four times faster.

**`fitch`**(*topology*, *alignment*)

Perform the Fitch algorithm on a given tree topology and associated alignment. Deprecated: Python implementation of the Fitch algorithm; see `fitch C++` module for a C++ implementation that is roughly four times faster.

### 13.2 Variables

Name	Description
<code>__package__</code>	<b>Value:</b> <code>'pylogeny'</code>

### 13.3 Class `profile_set`

Hold a set of `site_profile` profiles for an entire alignment.

#### 13.3.1 Methods

**`__init__`**(*self*, *alignment*)

**`__len__`**(*self*)

**`weight`**(*self*, *val*)

**`get`**(*self*, *val*)

**getForTaxa**(*self*, *val*, *tax*)

### 13.4 Class *site\_profile*

Consolidate the single-column alignment at a region into a set of components on the basis of similarity alone.

#### 13.4.1 Methods

**\_\_init\_\_**(*self*, *alignment*, *site*)

**\_\_eq\_\_**(*self*, *o*)

**\_\_ne\_\_**(*self*, *o*)

**\_\_str\_\_**(*self*)

## 14 Module *pylogeny.pll*

Wrap C extension for libpll library for use in natural Python.

### 14.1 Variables

Name	Description
<code>__package__</code>	Value: 'pylogeny'

### 14.2 Class *dataModel*

Encapsulating a phylogenetic tree (as topology) + corresponding alignment into a libpll-associated data structure. Allows for log-likelihood scoring of this model. **\*\*MUST BE CLOSED AFTER USE.\*\***

#### 14.2.1 Methods

<b><code>__init__</code></b> ( <i>self</i> , <i>topo</i> , <i>alignm</i> , <i>model=None</i> )
Initialize the data model and respective structures.  :param topo: A topology object. :type topo: :class: 'rearrangement.topology' :param alignm: A <i>phylipFriendlyAlignment</i> object. :type alignm: :class: 'alignment.phylipFriendlyAlignment'
<b><code>getNewickString</code></b> ( <i>self</i> )
Acquire the Newick string of the problem instance.
<b><code>getLogLikelihood</code></b> ( <i>self</i> )
Calculates log-likelihood using libpll.
<b><code>close</code></b> ( <i>self</i> )
If done with this particular problem. Frees associated memory.

### 14.3 Class *partitionModel*

A partition model intended for libpll.

### 14.3.1 Methods

<b><code>__init__</code></b> ( <i>self</i> , <i>ali</i> )
---

<b><code>getFileName</code></b> ( <i>self</i> )
---

Get the file name of the model file.
--------------------------------------

<b><code>createSimpleModel</code></b> ( <i>self</i> , <i>protein</i> , <i>pmodel</i> ='WAG')
--

Establish a simple model (e.g., one type).
--

<b><code>createModel</code></b> ( <i>self</i> , <i>models</i> , <i>partnames</i> , <i>ranges</i> )
--

Establish a more complex model.
---------------------------------

<b><code>close</code></b> ( <i>self</i> )
---

Delete file.
--------------

## 15 Module *pylogeny.rearrangement*

Phylogenetic tree structure encapsulation; allow rearrangement of said structure. Tree rearrangements inducing other topologies include Nearest Neighbor Interchange (NNI), Subtree Pruning and Regrafting (SPR), and Tree Bisection and Reconstruction (TBR). Each of these describe a transfer of one node in phylogenetic trees from one parent of a tree to a new parent. Respectively, these operators describe transformations that are subsets of those possible by the successive operator. For example, an NNI operator can perform transformations that are a subset of the transformations possible by the SPR operator.

### 15.1 Functions

**dup**(*topo*, *where*=None)

### 15.2 Variables

Name	Description
TYPE_NNI	<b>Value:</b> 2
TYPE_SPR	<b>Value:</b> 1
TYPE_TBR	<b>Value:</b> 3
__package__	<b>Value:</b> 'pylogeny'

### 15.3 Class *RearrangementError*



#### 15.3.1 Methods

**\_\_init\_\_**(*self*, *val*)

*x*.**\_\_init\_\_**(...) initializes *x*; see `help(type(x))` for signature

Overrides: `object.__init__` `exitit`(inherited documentation)



<b>__str__</b> ( <i>self</i> ) str( <i>x</i> ) Overrides: object.__str__ extit(inherited documentation)
---

### *Inherited from exceptions.Exception*

<b>__new__</b> ()
-------------------

### *Inherited from exceptions.BaseException*

<b>__delattr__</b> (), <b>__getattr__</b> (), <b>__getitem__</b> (), <b>__getslice__</b> (), <b>__reduce__</b> (), <b>__repr__</b> (), <b>__setattr__</b> (), <b>__setstate__</b> (), <b>__unicode__</b> ()
---

### *Inherited from object*

<b>__format__</b> (), <b>__hash__</b> (), <b>__reduce_ex__</b> (), <b>__sizeof__</b> (), <b>__subclasshook__</b> ()
---

## 15.3.2 Properties

Name	Description
<i>Inherited from exceptions.BaseException</i>	
args, message	
<i>Inherited from object</i>	
<b>__class__</b>	

## 15.4 Class rearrangement

Encapsulates a single rearrangement move of type SPR, NNI, ...

### 15.4.1 Methods

<b>__init__</b> ( <i>self</i> , <i>struct</i> , <i>type</i> , <i>targ</i> , <i>dest</i> )
Initialize by providing a pointer to a base topology, a target branch to be moved, and its destination.

<b>getType</b> ( <i>self</i> )
Get the type of movement.

<b>isNNI</b> ( <i>self</i> )
------------------------------

<b>isSPR</b> ( <i>self</i> )
------------------------------

<b>isTBR</b> ( <i>self</i> )
------------------------------

<b>toTopology</b> ( <i>self</i> )
-----------------------------------

Commit the actual move and return the topology.
---

<b>toNewick</b> ( <i>self</i> )
---------------------------------

Commit the move but do not create a new structure. Only retrieve resultant Newick string; will be more efficient.
---

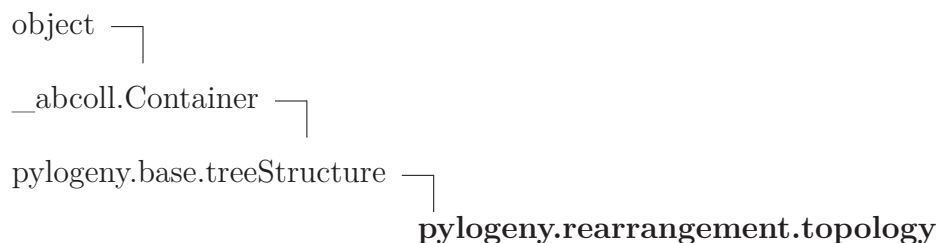
<b>toTree</b> ( <i>self</i> )
-------------------------------

Commit the move and transform to tree object.
---

<b>doMove</b> ( <i>self</i> )
-------------------------------

<b>__str__</b> ( <i>self</i> )
--------------------------------

## 15.5 Class topology



Encapsulate a tree topology, wrapping the newick tree structure. Is immutable.

### 15.5.1 Methods

<b>__init__</b> ( <i>self</i> , <i>t=None</i> , <i>rerootToLeaf=True</i> , <i>toLeaf=None</i> )
---

Initialize structure with a top-level internal node OR nothing.
---

Overrides: <code>object.__init__</code>
---

**rerootToLeaf**(*self*, *toleaf*=None)

PRIVATE: Reroots the given tree structure such that it is rooted nearest the lowest-order leaf.

**getBranches**(*self*)

**getLeaves**(*self*)

**getBipartitions**(*self*)

Get all bipartitions.

**getStrBipartitionFromBranch**(*self*, *br*)

Given a branch, return corresponding bipartition.

**getBranchFromStrBipartition**(*self*, *bip*)

Given a bipartition of taxa, return a branch that creates that partition of tree taxa.

**getBranchFromBipartition**(*self*, *bip*)

Given a bipartition object, return a branch that creates that partition of taxa.

**lockBranch**(*self*, *branch*)

Given a branch, lock it such that no transitions can ever occur across it.

**move**(*self*, *branch*, *destination*, *returnStruct*=True)

Move a branch and attach to a destination branch. Return new structure, or return merely the resultant Newick string.

**SPR**(*self*, *branch*, *destination*)

Perform an SPR move of a branch to a destination branch, creating a new node there. Returns a rearrangement structure (not the actual new structure) that can then be polled for the actual move; this is in order to save memory.

**NNI**(*self*, *branch*, *destination*)

Perform an NNI move of a branch to a destination, only if that destination branch is a parent's parent or a parent's sibling. Returns a rearrangement structure (not the actual new structure) that can then be polled for the actual move; this is in order to save memory.

**iterSPRForBranch**(*self*, *br*, *flip*=True)

Consider all valid SPR moves for a given branch in the topology and yield all possible rearrangements as a generator.

**allSPRForBranch**(*self*, *br*, *flip*=True)

Consider all valid SPR moves for a given branch in the topology and return all possible rearrangements.

**allSPR**(*self*)

Consider all valid SPR moves for a given topology and return all possible rearrangements.

**iterNNIForBranch**(*self*, *br*, *flip*=True)

Consider all valid NNI moves for a given branch in the topology and and yield all possible rearrangements as a generator.

**allNNIForBranch**(*self*, *br*, *flip*=True)

Consider all valid NNI moves for a given branch in the topology and return all possible rearrangements.

**allNNI**(*self*)

Consider all valid NNI moves for a given topology and return all possible rearrangements.

**allType**(*self*, *type*=1)

Consider all valid moves of a given rearrangement operator for a given topology. Uses a given rearrangement operator type defined in this module. For example, calling this function by providing TYPE\_NNI as the type will iterate over all NNI operations. By default, the type is TYPE\_SPR.

**iterTypeForBranch**(*self*, *br*, *type*=1, *flip*=True)

Iterate over all possible rearrangements for a branch using a given rearrangement operator type defined in this module. For example, calling this function by providing TYPE\_NNI as the type will iterate over all NNI operations. By default, the type is TYPE\_SPR.

**fromNewick**(*self*, *newickstr*)

Alias for parse().

<b>parse</b> ( <i>self</i> , <i>newickstr</i> )
---

Parse a newick string and assign the tree to this object. Cannot already be initialized with a tree.
--

<b>toNewick</b> ( <i>self</i> )
---------------------------------

Return the newick string of the tree.
---------------------------------------

<b>toUnrootedNewick</b> ( <i>self</i> )
---

Return the newick string of the tree as an unrooted topology with a multifurcating top-level node.
--

<b>toTree</b> ( <i>self</i> )
-------------------------------

Return the tree object for this topology.
---

<b>toUnrootedTree</b> ( <i>self</i> )
---------------------------------------

Return the tree object of the unrooted version of this topology.
--

<b>__str__</b> ( <i>self</i> )
--------------------------------

Return the newick string of the tree.
---------------------------------------

Overrides: object.__str__
---------------------------

**Inherited from *pylogeny.base.treeStructure*(Section 5.3)**

\_\_contains\_\_(), getAllLeaves(), getAllNodes(), getPostOrderTraversal(), getRoot(), leaves(), nodes(), postOrderTraversal()

**Inherited from *\_\_abcoll.Container***

\_\_subclasshook\_\_()

**Inherited from *object***

\_\_delattr\_\_(), \_\_format\_\_(), \_\_getattr\_\_(), \_\_hash\_\_(), \_\_new\_\_(), \_\_reduce\_\_(), \_\_reduce\_ex\_\_(), \_\_repr\_\_(), \_\_setattr\_\_(), \_\_sizeof\_\_()

### 15.5.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

**15.5.3 Class Variables**

Name	Description
__abstractmethods__, root	<i>Inherited from pylogeny.base.treeStructure (Section 5.3)</i>

## 16 Module *pylogeny.scoring*

Functions for phylogenetic tree goodness-of-fit scoring.

### 16.1 Functions

**beaglegetLogLikelihood**(*tree*, *alignment*)

Acquire log-likelihood via C++ library BEAGLE via use of pybeaglethon wrapper library. Currently uses HKY85 model.

:param tree: A tree object. :type tree: :class: 'tree.tree' :param alignment: An alignment object. :type alignment: :class: 'alignment.alignment' :returns: A floating point value.

**getLogLikelihood**(*tree*, *alignment*, *updateBranchLengths=True*)

Acquire log-likelihood via C library libpll.

:param tree: A tree object. :type tree: :class: 'tree.tree' :param alignment: An alignment object. :type alignment: :class: 'alignment.phylipFriendlyAlignment' :param updateBranchLengths: Whether or not to update the branch lengths in the provided tree with optimized ones. :returns: A floating point value.

**getParsimony**(*newick*, *alignment*)

Acquire parsimony via a C++ implementation.

:param newick: A New Hampshire (Newick) tree string. :param alignment: An alignment object. :type alignment: :class: 'alignment.alignment' :returns: An integer value.

**getParsimonyForTopology**(*topo*, *alignment*)

Acquire parsimony via a C++ implementation.

:param topo: A topology object. :type topo: :class: 'rearrangement.topology' :param alignment: An alignment object. :type alignment: :class: 'alignment.alignment' :returns: An integer value.

**getParsimonyFromProfiles**(*newick*, *profiles*)

Acquire parsimony via a C++ implementation.

:param newick: A New Hampshire (Newick) tree string. :param profiles: A set of profiles corresponding to an alignment. :type profiles: :class: 'parsimony.profile\_set' :returns: An integer value.

**getParsimonyFromProfilesForTopology**(*topology*, *profiles*)

Acquire parsimony via a C++ implementation.

:param topo: A topology object. :type topo: :class: 'rearrangement.topology'  
:param profiles: A set of profiles corresponding to an alignment. :type profiles: :class: 'parsimony.profile\_set' :returns: An integer value.

**16.2 Variables**

Name	Description
__package__	<b>Value:</b> 'pylogeny'



## 17 Module `pylogeny.tree`

Container definition for (phylogenetic) bifurcating or multifurcating trees defined using Newick strings, collections of them, and for splits of these trees.

### 17.1 Functions

<code>numberRootedTrees(<i>t</i>)</code>
--

<code>numberUnrootedTrees(<i>t</i>)</code>
--

### 17.2 Variables

Name	Description
<code>__package__</code>	Value: <code>'pylogeny'</code>

### 17.3 Class tree

```

object └─
          pylogeny.tree.tree

```

Defines a single (phylogenetic) tree by newick string; can possess other metadata.

#### 17.3.1 Methods

<code>__init__(<i>self</i>, <i>newi</i>='', <i>check</i>=False)</code>
--

If enabled, "check" will force the structure to reroot the given Newick string tree to a lowest-order leaf in order to ensure a consistent Newick string among any duplicate topologies.

Overrides: `object.__init__`

<code>getName(<i>self</i>)</code>
-----------------------------------

<code>setName(<i>self</i>, <i>n</i>)</code>
---

<code>getScore(<i>self</i>)</code>
------------------------------------

**setScore**(*self*, *s*)

**getOrigin**(*self*)

**setOrigin**(*self*, *o*)

Set the "origin" or specification of where this tree was acquired or constructed from; a string.

**getNewick**(*self*)

**toNewick**(*self*)

**updateNewick**(*self*, *n*, *reroot=False*)

Update the contained Newick string only as long as the structure obtained (after rerooting, which is an optional parameter) is identical to the contained structure.

**getStructure**(*self*)

Returns the tree's "structure", a Newick string without any branch lengths.

**getRerootedNoBranchLengthNewick**(*self*)

See getStructure().

**getSimpleNewick**(*self*)

Return a Newick string with all taxa name replaced with successive integers.

**toTopology**(*self*)

Return a rearrangement.topology instance for this tree to allow for rearrangement of the actual structure of the tree.

**\_\_eq\_\_**(*self*, *o*)

**\_\_ne\_\_**(*self*, *o*)

**\_\_str\_\_**(*self*)

str(x)

Overrides: object.\_\_str\_\_ extit(inherited documentation)

***Inherited from object***

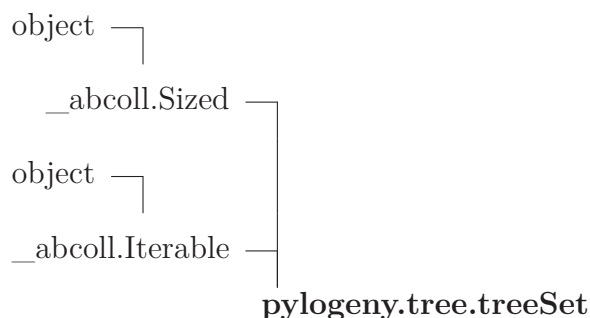
```

__delattr__(), __format__(), __getattr__(), __hash__(), __new__(),
__reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(),
__subclasshook__()

```

**17.3.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

**17.4 Class *treeSet*****Known Subclasses:** `pylogeny.landscape.landscape`

Represents an ordered, disorganized collection of trees that do not necessarily comprise a combinatorial space.

**17.4.1 Methods**

```
__init__(self)
```

`x.__init__(...)` initializes `x`; see `help(type(x))` for signature

Overrides: `object.__init__` `extit`(inherited documentation)

```
addTree(self, tr)
```

Add a tree object to the collection.

```
addTreeByNewick(self, newick)
```

Add a tree to the structure by Newick string.

**removeTree**(*self*, *tr*)

Remove a tree object from the collection if present.

**indexOf**(*self*, *tr*)

Acquire the index in this collection of a tree object. Returns -1 if not found.

**\_\_getitem\_\_**(*self*, *i*)

**\_\_setitem\_\_**(*self*, *i*, *o*)

**\_\_len\_\_**(*self*)

Overrides: *\_\_abcoll.Sized.\_\_len\_\_*

**\_\_iter\_\_**(*self*)

Overrides: *\_\_abcoll.Iterable.\_\_iter\_\_*

**iterTrees**(*self*)

Iterate over all trees found in this set.

**toTreeFile**(*self*, *fout*)

Output this landscape as a series of trees, separated by newlines, as a text file saved at the given path.

**fromTreeFile**(*fin*)

Acquire a file where newlines separate Newick strings, and create an instance of *treeSet* from those trees.

**\_\_str\_\_**(*self*)

*str*(*x*)

Overrides: *object.\_\_str\_\_* *\_\_extit\_\_*(inherited documentation)

**Inherited from *\_\_abcoll.Sized***

*\_\_subclasshook\_\_*()

**Inherited from *object***

*\_\_delattr\_\_*(), *\_\_format\_\_*(), *\_\_getattr\_\_*(), *\_\_hash\_\_*(), *\_\_new\_\_*(),  
*\_\_reduce\_\_*(), *\_\_reduce\_ex\_\_*(), *\_\_repr\_\_*(), *\_\_setattr\_\_*(), *\_\_sizeof\_\_*()

### 17.4.2 Properties

Name	Description
<i>Inherited from object</i> __class__	

### 17.4.3 Class Variables

Name	Description
__abstractmethods__	<b>Value:</b> frozenset([])

## 17.5 Class bipartition

object └─  
          **pylogeny.tree.bipartition**

A tree bipartition. Requires a tree topology. Using the term borrowed from nomenclature of a bipartite graph, a bipartition for a phylogenetic tree coincides with the definition of two disjoint sets U and V . A branch in a phylogenetic tree defines a single bipartition that divides the tree into two disjoint sets U and V . The set U comprises all of the children leaf of the subtree associated with that branch. The set V contains the rest of the leaves or taxa in the tree.

### 17.5.1 Methods

<b>__init__</b> ( <i>self</i> , <i>topol</i> , <i>bra</i> =None) <hr/> Construct a bipartition from a branch in a topology.  :param <i>topol</i> : A topology. :type <i>topol</i> : :class: 'rearrangement.topology' :param <i>bra</i> : An optional argument; can still acquire a bipartition from a string. :type <i>bra</i> : :class: 'newick.branch'  Overrides: object.__init__
<b>__hash__</b> ( <i>self</i> )  hash( <i>x</i> )  Overrides: object.__hash__ extit(inherited documentation)
<b>__eq__</b> ( <i>self</i> , <i>o</i> )

**\_\_ne\_\_**(*self*, *o*)

**fromStringRepresentation**(*self*, *st*)

Acquire all component elements from a string representation of a bipartition.

:param st: A string representation from a :class:'.bipartition' object.

**getBranch**(*self*)

Get branch corresponding to this bipartition.

:returns: :class:'.newick.branch'

**getBranchIndex**(*self*)

Return an index of the branch with respect to a post order traversal of the topology.

**getStringRepresentation**(*self*)

Get the string representation corresponding to this bipartition.

**getShortStringRepresentation**(*self*)

Get the shorter string representation corresponding to this bipartition.

**getShortStringMappings**(*self*)

Get the mapping of symbols from taxa names for the shorter string representation.

**getBranchListRepresentation**(*self*)

Get the tuple of lists of branches that represent this bipartition.

**getSPRRearrangements**(*self*)

Return the set of all scores related to this bipartition.

**getSPRScores**(*self*, *ls*, *node=None*)

Given a landscape, return all possible scores, not actively performing scoring if not done.

**getMedianSPRScore**(*self*, *ls*, *node=None*)

Given a landscape, return the median SPR score.

<b>getBestSPRScore</b> ( <i>self</i> , <i>ls</i> , <i>node=None</i> )
---

Given a landscape, return the best SPR score.
---

***Inherited from object***

```
__delattr__(), __format__(), __getattr__(), __new__(), __reduce__(),
__reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __sub-
classhook__()
```

**17.5.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

## Index

- pylogeny (*package*), 2–3
  - pylogeny.\_\_version\_\_ (*module*), 5
  - pylogeny.alignment (*module*), 6–10
    - pylogeny.alignment.alignment (*class*), 6–8
    - pylogeny.alignment.phylipFriendlyAlignment (*class*), 8–10
  - pylogeny.base (*module*), 11–20
    - pylogeny.base.longest\_common\_substring (*function*), 11
    - pylogeny.base.patriciaTree (*class*), 18–20
    - pylogeny.base.treeBranch (*class*), 14–15
    - pylogeny.base.treeNode (*class*), 12–14
    - pylogeny.base.treeStructure (*class*), 11–12
    - pylogeny.base.trie (*class*), 16–18
    - pylogeny.base.trieNode (*class*), 15–16
  - pylogeny.database (*module*), 21–32
    - pylogeny.database.database (*class*), 27–29
    - pylogeny.database.DatabaseLandscape (*class*), 21–22
    - pylogeny.database.SQLDatabase (*class*), 29–31
    - pylogeny.database.SQLExhaustiveLandscape (*class*), 22–25
    - pylogeny.database.SQLiteDatabase (*class*), 31–32
    - pylogeny.database.SQLiteLandscape (*class*), 25–27
  - pylogeny.executable (*module*), 33–41
    - pylogeny.executable.aTemporaryDirectory (*class*), 33–34
    - pylogeny.executable.consel (*class*), 36–37
    - pylogeny.executable.executable (*class*), 34–35
    - pylogeny.executable.exeExists (*function*), 33
    - pylogeny.executable.fasttree (*class*), 37–38
    - pylogeny.executable.raxml (*class*), 38–39
    - pylogeny.executable.rspr (*class*), 39–41
    - pylogeny.executable.treepuzzle (*class*), 35–36
  - pylogeny.heuristic (*module*), 42–49
    - pylogeny.heuristic.heuristic (*class*), 42–43
    - pylogeny.heuristic.likelihoodGreedy (*class*), 45–46
    - pylogeny.heuristic.parsimonyGreedy (*class*), 44–45
    - pylogeny.heuristic.phylogeneticLinearHeuristic (*class*), 43–44
    - pylogeny.heuristic.RAxMLIdentify (*class*), 47–49
    - pylogeny.heuristic.smoothGreedy (*class*), 46–47
  - pylogeny.JSONWriter (*module*), 3–4
    - pylogeny.JSONWriter.JSONWriter (*class*), 3–4
  - pylogeny.landscape (*module*), 50–61
    - pylogeny.landscape.graph (*class*), 50–53
    - pylogeny.landscape.landscape (*class*), 53–58
    - pylogeny.landscape.vertex (*class*), 58–61
  - pylogeny.landscapeWriter (*module*), 62–63
    - pylogeny.landscapeWriter.landscapeParser (*class*), 63
    - pylogeny.landscapeWriter.landscapeWriter (*class*), 62–63
  - pylogeny.model (*module*), 64–66
    - pylogeny.model.DiscreteStateModel (*class*), 65–66
    - pylogeny.model.PhyloModelError (*class*), 64–65
  - pylogeny.newick (*module*), 67–71
    - pylogeny.newick.assignParents (*function*), 67
    - pylogeny.newick.branch (*class*), 70–71



- pylogeny.newick.getAllBranches (*function*), 67
- pylogeny.newick.getBalancingBracket (*function*), 67
- pylogeny.newick.getBranchLength (*function*), 67
- pylogeny.newick.getLeafName (*function*), 68
- pylogeny.newick.invertAlongPathToNode (*function*), 67
- pylogeny.newick.isSibling (*function*), 67
- pylogeny.newick.newickParser (*class*), 71
- pylogeny.newick.node (*class*), 69–70
- pylogeny.newick.parseNewick (*function*), 68
- pylogeny.newick.ParsingError (*class*), 68–69
- pylogeny.newick.removeBranchLengths (*function*), 67
- pylogeny.newick.removeUnaryInternalNodes (*function*), 67
- pylogeny.newick.shuffleLeaves (*function*), 67
- pylogeny.parsimony (*module*), 72–73
  - pylogeny.parsimony.fitch (*function*), 72
  - pylogeny.parsimony.fitch\_cost (*function*), 72
  - pylogeny.parsimony.profile\_set (*class*), 72–73
  - pylogeny.parsimony.site\_profile (*class*), 73
- pylogeny.pll (*module*), 74–75
  - pylogeny.pll.dataModel (*class*), 74
  - pylogeny.pll.partitionModel (*class*), 74–75
- pylogeny.rearrangement (*module*), 76–82
  - pylogeny.rearrangement.dup (*function*), 76
  - pylogeny.rearrangement.rearrangement (*class*), 77–78
  - pylogeny.rearrangement.RearrangementError (*class*), 76–77
  - pylogeny.rearrangement.topology (*class*), 78–82
- pylogeny.scoring (*module*), 83–84
  - pylogeny.scoring.beaglegetLogLikelihood (*function*), 83
  - pylogeny.scoring.getLogLikelihood (*function*), 83
  - pylogeny.scoring.getParsimony (*function*), 83
  - pylogeny.scoring.getParsimonyForTopology (*function*), 83
  - pylogeny.scoring.getParsimonyFromProfiles (*function*), 83
  - pylogeny.scoring.getParsimonyFromProfilesForTopology (*function*), 84
- pylogeny.tree (*module*), 85–91
  - pylogeny.tree.bipartition (*class*), 89–91
  - pylogeny.tree.numberRootedTrees (*function*), 85
  - pylogeny.tree.numberUnrootedTrees (*function*), 85
  - pylogeny.tree.tree (*class*), 85–87
  - pylogeny.tree.treeSet (*class*), 87–89