```
Help on module nltk.tree in nltk:
                                                                                    pop(self, v=None)
NAME
  nltk.tree
                                                                                   remove(self, v)
FILE
                                                                                   reverse(self)
  /usr/local/lib/python2.7/dist-packages/nltk/tree.py
                                                                                   sort(self)
DESCRIPTION
  Class for representing hierarchical language structures, such as
  syntax trees and morphological trees.
                                                                                    Data descriptors inherited from ImmutableTree:
CLASSES
                                                                                   node
    _builtin__.list(__builtin__.object)
                                                                                      Get the node value
     Tree
       ImmutableTree
                                                                                    Methods inherited from MultiParentedTree:
          ImmutableMultiParentedTree(ImmutableTree,
MultiParentedTree)
          ImmutableParentedTree(ImmutableTree, ParentedTree)
                                                                                   left_siblings(self)
                                                                                      A list of all left siblings of this tree, in any of its parent
          ImmutableProbabilisticTree(ImmutableTree,
nltk.probability.ProbabilisticMixIn)
                                                                                      trees. A tree may be its own left sibling if it is used as
       ProbabilisticTree(Tree, nltk.probability.ProbabilisticMixIn)
                                                                                      multiple contiguous children of the same parent. A tree may
  builtin .obiect
                                                                                      appear multiple times in this list if it is the left sibling
     nltk.probability.ProbabilisticMixIn
                                                                                      of this tree with respect to multiple parents.
  AbstractParentedTree(Tree)
     MultiParentedTree
                                                                                      :type: list(MultiParentedTree)
     ParentedTree
                                                                                   parent indices(self, parent)
                                                                                      Return a list of the indices where this tree occurs as a child
  class ImmutableMultiParentedTree(ImmutableTree,
MultiParentedTree)
                                                                                      of "parent". If this child does not occur as a child of
    Method resolution order:
                                                                                      "parent", then the empty list is returned. The following is
       ImmutableMultiParentedTree
                                                                                      always true::
       ImmutableTree
       MultiParentedTree
                                                                                       for parent index in ptree.parent indices(parent):
       AbstractParentedTree
                                                                                          parent[parent_index] is ptree
       Tree
         builtin__.list
                                                                                   parents(self)
                                                                                      The set of parents of this tree. If this tree has no parents,
         _builtin___.object
                                                                                      then "parents" is the empty set. To check if a tree is used
     Methods inherited from ImmutableTree:
                                                                                      as multiple children of the same parent, use the
                                                                                      "parent indices()" method.
     __delitem__(self, index)
                                                                                      :type: list(MultiParentedTree)
     __delslice__(self, i, j)
                                                                                   right_siblings(self)
     __hash__(self)
                                                                                      A list of all right siblings of this tree, in any of its parent
                                                                                      trees. A tree may be its own right sibling if it is used as
                                                                                      multiple contiguous children of the same parent. A tree may
     __iadd__(self, other)
                                                                                      appear multiple times in this list if it is the right sibling
                                                                                      of this tree with respect to multiple parents.
     __imul__(self, other)
     __init__(self, node_or_str, children=None)
                                                                                      :type: list(MultiParentedTree)
     __setitem__(self, index, value)
                                                                                   roots(self)
                                                                                      The set of all roots of this tree. This set is formed by
                                                                                      tracing all possible parent paths until trees with no parents
     __setslice__(self, i, j, value)
                                                                                      are found.
     append(self, v)
                                                                                      :type: list(MultiParentedTree)
     extend(self, v)
```

```
treepositions(self, root)
                                                                                      :type horzMarkov: int | None
       Return a list of all tree positions that can be used to reach
                                                                                      :param vertMarkov: Markov order for parent smoothing (0
       this multi-parented tree starting from "root". I.e., the
                                                                              (default) = no vertical annotation)
       following is always true::
                                                                                      :type vertMarkov: int | None
                                                                                      :param childChar: A string used in construction of the artificial
                                                                              nodes, separating the head of the
        for treepos in ptree.treepositions(root):
           root[treepos] is ptree
                                                                                                 original subtree from the child nodes that have yet
                                                                              to be expanded (default = "|")
                                                                                      :type childChar: str
    Methods inherited from AbstractParentedTree:
                                                                                      :param parentChar: A string used to separate the node
                                                                              representation from its vertical annotation
                                                                                      :type parentChar: str
      _getslice__(self, start, stop)
    insert(self, index, child)
                                                                                   collapse_unary(self, collapsePOS=False, collapseRoot=False,
                                                                              joinChar='+')
                                                                                      Collapse subtrees with a single child (ie. unary productions)
                                                                                      into a new non-terminal (Tree node) joined by 'joinChar'.
    Methods inherited from Tree:
                                                                                      This is useful when working with algorithms that do not allow
     __add__(self, v)
                                                                                      unary productions, and completely removing the unary
                                                                              productions
     __eq__(self, other)
                                                                                     would require loss of useful information. The Tree is modified
                                                                                      directly (since it is passed by reference) and no value is
                                                                              returned.
     __ge__(self, other)
                                                                                      :param collapsePOS: 'False' (default) will not collapse the
     __getitem__(self, index)
                                                                              parent of leaf nodes (ie.
                                                                                                   Part-of-Speech tags) since they are always unary
     __gt__(self, other)
                                                                              productions
                                                                                      :type collapsePOS: bool
     __le__(self, other)
                                                                                      :param collapseRoot: 'False' (default) will not modify the root
     __lt__(self, other)
                                                                              production
                                                                                                   if it is unary. For the Penn WSJ treebank corpus,
                                                                              this corresponds
     mul (self, v)
                                                                                                   to the TOP -> productions.
     __ne__(self, other)
                                                                                      :type collapseRoot: bool
                                                                                      :param joinChar: A string used to connect collapsed node
                                                                              values (default = "+")
     __radd__(self, v)
                                                                                      :type joinChar: str
     __repr__(self)
                                                                                   copy(self, deep=False)
     __rmul__(self, v)
                                                                                   draw(self)
     __str__(self)
                                                                                      Open a new window containing a graphical diagram of this
                                                                              tree.
    chomsky_normal_form(self, factor='right', horzMarkov=None,
vertMarkov=0, childChar='|', parentChar='^')
                                                                                   flatten(self)
       This method can modify a tree in three ways:
                                                                                     Return a flat version of the tree, with all non-root non-
                                                                              terminals removed.
         1. Convert a tree into its Chomsky Normal Form (CNF)
          equivalent -- Every subtree has either two non-terminals
                                                                                        >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
          or one terminal as its children. This process requires
                                                                              (D the) (N cat)))")
                                                                                        >>> print t.flatten()
          the creation of more "artificial" non-terminal nodes.
         2. Markov (vertical) smoothing of children in new artificial
                                                                                        (S the dog chased the cat)
          nodes
         3. Horizontal (parent) annotation of nodes
                                                                                      :return: a tree consisting of this tree's root connected directly
                                                                              to
        :param factor: Right or left factoring method (default = "right")
                                                                                        its leaves, omitting all intervening non-terminal nodes.
       :type factor: str = [left|right]
                                                                                      :rtype: Tree
        :param horzMarkov: Markov order for sibling smoothing in
artificial nodes (None (default) = include all siblings)
                                                                                   freeze(self, leaf_freezer=None)
```

```
:param margin: The right margin at which to do line-wrapping.
     height(self)
                                                                                        :type margin: int
                                                                                        :param indent: The indentation level at which printing
       Return the height of the tree.
                                                                                           begins. This number is used to decide how far to indent
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                           subsequent lines.
(D the) (N cat))))")
                                                                                        :type indent: int
          >>> t.height()
                                                                                        :param nodesep: A string that is used to separate the node
                                                                                           from the children. E.g., the default value ``':'`` gives
                                                                                           trees like ``(S: (NP: I) (VP: (V: saw) (NP: it)))``.
          >>> print t[0,0]
          (D the)
          >>> t[0,0].height()
                                                                                     pprint latex gtree(self)
                                                                                        Returns a representation of the tree compatible with the
                                                                                        LaTeX gtree package. This consists of the string "\Tree"
       :return: The height of this tree. The height of a tree
                                                                                        followed by the parse tree represented in bracketed notation.
          containing no children is 1; the height of a tree
          containing only leaves is 2; and the height of any other
                                                                                        For example, the following result was generated from a parse
          tree is one plus the maximum of its children's
                                                                                tree of
                                                                                        the sentence "The announcement astounded us":::
          heights.
       :rtype: int
                                                                                         \Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
     leaf_treeposition(self, index)
                                                                                            [.l' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
       :return: The tree position of the ``index``-th leaf in this
          tree. I.e., if "tp=self.leaf treeposition(i)", then
                                                                                        See http://www.ling.upenn.edu/advice/latex.html for the
           ``self[tp]==self.leaves()[i]``.
                                                                                LaTeX
                                                                                        style file for the qtree package.
       :raise IndexError: If this tree contains fewer than ``index+1``
          leaves, or if ``index<0``.
                                                                                        :return: A latex gtree representation of this tree.
                                                                                        :rtype: str
     leaves(self)
       Return the leaves of the tree.
                                                                                     productions(self)
                                                                                        Generate the productions that correspond to the non-terminal
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                nodes of the tree.
                                                                                        For each subtree of the form (P: C1 C2 ... Cn) this produces a
(D the) (N cat))))")
                                                                                production of the
          >>> t.leaves()
                                                                                        form P -> C1 C2 ... Cn.
          ['the', 'dog', 'chased', 'the', 'cat']
                                                                                           >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
       :return: a list containing this tree's leaves.
          The order reflects the order of the
                                                                                (D the) (N cat))))")
          leaves in the tree's hierarchical structure.
                                                                                           >>> t.productions()
                                                                                           [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP,
       :rtype: list
                                                                                V -> 'chased'.
                                                                                           NP -> D N, D -> 'the', N -> 'cat']
     pos(self)
       Return a sequence of pos-tagged words extracted from the
                                                                                        :rtype: list(Production)
tree.
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                     subtrees(self, filter=None)
                                                                                        Generate all the subtrees of this tree, optionally restricted
(D the) (N cat))))")
                                                                                        to trees matching the filter function.
          >>> t.pos()
          [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
                                                                                           >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
       :return: a list of tuples containing leaves and pre-terminals
                                                                                (D the) (N cat))))")
(part-of-speech tags).
                                                                                           >>> for s in t.subtrees(lambda t: t.height() == 2):
          The order reflects the order of the leaves in the tree's
                                                                                           ... print s
hierarchical structure.
                                                                                           (D the)
       :rtype: list(tuple)
                                                                                           (N dog)
                                                                                           (V chased)
    pprint(self, margin=70, indent=0, nodesep=", parens='()',
                                                                                           (D the)
quotes=False)
                                                                                           (N cat)
       :return: A pretty-printed string representation of this tree.
```

:type filter: function

:rtvpe: str

```
:param filter: the function to filter all local trees
                                                                                       are applied to the substrings of "s" corresponding to
                                                                                       nodes and leaves (respectively) to obtain the values for
    treeposition spanning leaves(self, start, end)
                                                                                       those nodes and leaves. They should have the following
       :return: The tree position of the lowest descendant of this
                                                                                       signature:
          tree that dominates ``self.leaves()[start:end]``.
       :raise ValueError: if ``end <= start``
                                                                                         parse node(str) -> value
    un_chomsky_normal_form(self, expandUnary=True,
                                                                                       For example, these functions could be used to parse nodes
childChar='|', parentChar='^', unaryChar='+')
                                                                                       and leaves whose values should be some type other than
       This method modifies the tree in three ways:
                                                                                       string (such as ``FeatStruct``).
                                                                                       Note that by default, node strings and leaf strings are
        1. Transforms a tree in Chomsky Normal Form back to its
                                                                                       delimited by whitespace and brackets; to override this
                                                                                       default, use the "node_pattern" and "leaf_pattern"
          original structure (branching greater than two)
        2. Removes any parent annotation (if it exists)
                                                                                       arguments.
        3. (optional) expands unary subtrees (if previously
          collapsed with collapseUnary(...))
                                                                                     :type node pattern: str
                                                                                     :type leaf_pattern: str
       :param expandUnary: Flag to expand unary or not (default =
                                                                                    :param node_pattern, leaf_pattern: Regular expression
True)
                                                                             patterns
       :type expandUnary: bool
                                                                                       used to find node and leaf substrings in "s". By
       :param childChar: A string separating the head node from its
                                                                                       default, both nodes patterns are defined to match any
children in an artificial node (default = "|")
                                                                                       sequence of non-whitespace non-bracket characters.
       :type childChar: str
       :param parentChar: A sting separating the node label from its
                                                                                     :type remove_empty_top_bracketing: bool
parent annotation (default = "^")
                                                                                     :param remove_empty_top_bracketing: If the resulting tree
       :type parentChar: str
                                                                             has
       :param unaryChar: A string joining two non-terminals in a
                                                                                       an empty node label, and is length one, then return its
unary production (default = "+")
                                                                                       single child instead. This is useful for treebank trees,
       :type unaryChar: str
                                                                                       which sometimes contain an extra level of bracketing.
                                                                                     :return: A tree corresponding to the string representation ``s``.
                                                                                       If this class method is called using a subclass of Tree,
     Class methods inherited from Tree:
                                                                                       then it will return a tree of that type.
                                                                                    :rtype: Tree
    convert(cls, tree) from builtin .type
       Convert a tree between different subtypes of Tree. "cls"
determines
       which class will be used to encode the new tree.
                                                                                  Data descriptors inherited from Tree:
       :type tree: Tree
       :param tree: The tree that should be converted.
                                                                                    dictionary for instance variables (if defined)
       :return: The new Tree.
                                                                                    weakref
    parse(cls, s, brackets='()', parse_node=None, parse_leaf=None,
                                                                                    list of weak references to the object (if defined)
node_pattern=None, leaf_pattern=None,
remove_empty_top_bracketing=False) from __builtin__.type
       Parse a bracketed tree string and return the resulting tree.
                                                                                  Methods inherited from __builtin__.list:
       Trees are represented as nested brackettings, such as::
                                                                                    _contains__(...)
        (S (NP (NNP John)) (VP (V runs)))
                                                                                    x.__contains__(y) <==> y in x
       :type s: str
                                                                                    _getattribute__(...)
                                                                                    x.__getattribute__('name') <==> x.name
       :param s: The string to parse
       :type brackets: str (length=2)
                                                                                  __iter__(...)
       :param brackets: The bracket characters used to mark the
                                                                                    x.__iter__() <==> iter(x)
          beginning and end of trees and subtrees.
                                                                                  __len__(...)
                                                                                    x.__len__() <==> len(x)
       :type parse_node: function
       :type parse leaf: function
       :param parse_node, parse_leaf: If specified, these functions
                                                                                    reversed (...)
```

!	Lieverseu() return a reverse iterator over the list	
	sizeof()	Data descriptors inherited from ImmutableTree:
į	Lsizeof() size of L in memory, in bytes	node node Get the node value
	count()	Get the hode value
ا valu	L.count(value) -> integer return number of occurrences of le	 Methods inherited from ParentedTree:
	index()	l left_sibling(self)
ا valu	L.index(value, [start, [stop]]) -> integer return first index of le.	The left sibling of this tree, or None if it has none.
	Raises ValueError if the value is not present.	parent(self) The parent of this tree, or None if it has no parent.
	Data and other attributes inherited frombuiltinlist:	parent_index(self)
	new = <built-in methodnew="" object="" of="" type=""></built-in>	The index of this tree in its parent. I.e., '`ptree.parent()[ptree.parent_index()] is ptree``. Note that
İ	Tnew(S,) -> a new object with type S, a subtype of T	``ptree.parent_index()`` is not necessarily equal to
С	lass ImmutableParentedTree(ImmutableTree, ParentedTree)	``ptree.parent.index(ptree)``, since the ``index()`` method returns the first child that is equal to its argument.
	Method resolution order: ImmutableParentedTree	
	Immutable Parented Tree	right_sibling(self) The right sibling of this tree, or None if it has none.
į	ParentedTree	
	AbstractParentedTree Tree	root(self) The root of this tree. I.e., the unique ancestor of this tree
	builtinlist	whose parent is None. If ``ptree.parent()`` is None, then
į	builtinobject	``ptree`` is its own root.
 	Methods inherited from ImmutableTree:	treeposition(self) The tree position of this tree, relative to the root of the
j	delitem(self, index)	tree. I.e., ``ptree.root[ptree.treeposition] is ptree``.
	delslice(self, i, j)	
	hash(self)	Methods inherited from AbstractParentedTree:
		 getslice(self, start, stop)
	iadd(self, other)	 insert(self, index, child)
 	imul(self, other)	
	init(self, node_or_str, children=None)	Methods inherited from Tree:
	setitem(self, index, value)	add(self, v)
 	setslice(self, i, j, value)	eq(self, other)
	append(self, v)	ge(self, other)
	extend(self, v)	 getitem(self, index)
 	pop(self, v=None)	l gt(self, other)
 	remove(self, v)	l le(self, other)
 	reverse(self)	lt(self, other)
	sort(self)	 mul(self, v)

```
__ne__(self, other)
                                                                                      :type collapseRoot: bool
                                                                                       :param joinChar: A string used to connect collapsed node
                                                                               values (default = "+")
     radd (self, v)
                                                                                      :type joinChar: str
     __repr__(self)
                                                                                    copy(self, deep=False)
     __rmul__(self, v)
                                                                                    draw(self)
                                                                                      Open a new window containing a graphical diagram of this
     str (self)
                                                                               tree.
     chomsky normal form(self, factor='right', horzMarkov=None,
vertMarkov=0, childChar='|', parentChar='^')
                                                                                   flatten(self)
       This method can modify a tree in three ways:
                                                                                      Return a flat version of the tree, with all non-root non-
                                                                               terminals removed.
         1. Convert a tree into its Chomsky Normal Form (CNF)
           equivalent -- Every subtree has either two non-terminals
                                                                                         >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
           or one terminal as its children. This process requires
                                                                               (D the) (N cat))))")
           the creation of more "artificial" non-terminal nodes.
                                                                                         >>> print t.flatten()
         2. Markov (vertical) smoothing of children in new artificial
                                                                                         (S the dog chased the cat)
           nodes
         3. Horizontal (parent) annotation of nodes
                                                                                      :return: a tree consisting of this tree's root connected directly
                                                                               to
        :param factor: Right or left factoring method (default = "right")
                                                                                         its leaves, omitting all intervening non-terminal nodes.
        :type factor: str = [left|right]
                                                                                       :rtvpe: Tree
        :param horzMarkov: Markov order for sibling smoothing in
artificial nodes (None (default) = include all siblings)
                                                                                    freeze(self, leaf_freezer=None)
       :type horzMarkov: int | None
        :param vertMarkov: Markov order for parent smoothing (0
                                                                                    height(self)
(default) = no vertical annotation)
                                                                                      Return the height of the tree.
        :type vertMarkov: int | None
        :param childChar: A string used in construction of the artificial
                                                                                         >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
nodes, separating the head of the
                                                                               (D the) (N cat))))")
                   original subtree from the child nodes that have yet
                                                                                         >>> t.height()
to be expanded (default = "|")
                                                                                         >>> print t[0,0]
        :type childChar: str
       :param parentChar: A string used to separate the node
                                                                                         (D the)
representation from its vertical annotation
                                                                                         >>> t[0,0].height()
        :type parentChar: str
    collapse unary(self, collapsePOS=False, collapseRoot=False,
                                                                                      :return: The height of this tree. The height of a tree
                                                                                         containing no children is 1; the height of a tree
joinChar='+')
       Collapse subtrees with a single child (ie. unary productions)
                                                                                         containing only leaves is 2; and the height of any other
       into a new non-terminal (Tree node) joined by 'joinChar'.
                                                                                         tree is one plus the maximum of its children's
       This is useful when working with algorithms that do not allow
                                                                                         heights.
       unary productions, and completely removing the unary
                                                                                      :rtype: int
productions
       would require loss of useful information. The Tree is modified
                                                                                    leaf treeposition(self. index)
       directly (since it is passed by reference) and no value is
                                                                                       :return: The tree position of the ``index``-th leaf in this
                                                                                         tree. I.e., if ``tp=self.leaf_treeposition(i)``, then
returned.
                                                                                          `self[tp]==self.leaves()[i]``.
        :param collapsePOS: 'False' (default) will not collapse the
parent of leaf nodes (ie.
                                                                                      :raise IndexError: If this tree contains fewer than ``index+1``
                                                                                         leaves, or if ``index<0``.
                    Part-of-Speech tags) since they are always unary
productions
        :type collapsePOS: bool
                                                                                    leaves(self)
       :param collapseRoot: 'False' (default) will not modify the root
                                                                                      Return the leaves of the tree.
production
                     if it is unary. For the Penn WSJ treebank corpus,
                                                                                         >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
this corresponds
                                                                               (D the) (N cat))))")
                     to the TOP -> productions.
                                                                                         >>> t.leaves()
```

```
['the', 'dog', 'chased', 'the', 'cat']
                                                                                                                                      form P -> C1 C2 ... Cn.
                                                                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
            :return: a list containing this tree's leaves.
               The order reflects the order of the
                                                                                                                          (D the) (N cat)))")
               leaves in the tree's hierarchical structure.
                                                                                                                                          >>> t.productions()
                                                                                                                                          [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP,
            :rtype: list
                                                                                                                          V -> 'chased'.
                                                                                                                                          NP -> D N, D -> 'the', N -> 'cat']
       pos(self)
            Return a sequence of pos-tagged words extracted from the
tree.
                                                                                                                                       :rtype: list(Production)
               >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                                  subtrees(self, filter=None)
(D the) (N cat))))")
                                                                                                                                       Generate all the subtrees of this tree, optionally restricted
               >>> t.pos()
                                                                                                                                       to trees matching the filter function.
               [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
                                                                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
            :return: a list of tuples containing leaves and pre-terminals
                                                                                                                          (D the) (N cat))))")
(part-of-speech tags).
                                                                                                                                          >>> for s in t.subtrees(lambda t: t.height() == 2):
               The order reflects the order of the leaves in the tree's
                                                                                                                                                  print s
hierarchical structure.
                                                                                                                                           (D the)
            :rtype: list(tuple)
                                                                                                                                           (N dog)
                                                                                                                                           (V chased)
       pprint(self, margin=70, indent=0, nodesep=", parens='()',
                                                                                                                                           (D the)
quotes=False)
                                                                                                                                           (N cat)
            :return: A pretty-printed string representation of this tree.
                                                                                                                                       :type filter: function
            :rtvpe: str
            :param margin: The right margin at which to do line-wrapping.
                                                                                                                                       :param filter: the function to filter all local trees
            :type margin: int
            :param indent: The indentation level at which printing
                                                                                                                                  treeposition_spanning_leaves(self, start, end)
               begins. This number is used to decide how far to indent
                                                                                                                                       :return: The tree position of the lowest descendant of this
               subsequent lines.
                                                                                                                                          tree that dominates "self.leaves()[start:end]".
            :type indent: int
                                                                                                                                       :raise ValueError: if ``end <= start``
            :param nodesep: A string that is used to separate the node
               from the children. E.g., the default value ":" gives
                                                                                                                                  treepositions(self, order='preorder')
               trees like ``(S: (NP: I) (VP: (V: saw) (NP: it)))``.
                                                                                                                                           >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                          (D the) (N cat)))")
                                                                                                                                          >>> t.treepositions() # doctest: +ELLIPSIS
       pprint_latex_qtree(self)
            Returns a representation of the tree compatible with the
                                                                                                                                          [(), (0,), (0, 0), (0, 0, 0), (0, 1), (0, 1, 0), (1,), (1, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (
            LaTeX qtree package. This consists of the string "\Tree"
                                                                                                                          0), ...]
            followed by the parse tree represented in bracketed notation.
                                                                                                                                          >>> for pos in t.treepositions('leaves'):
                                                                                                                                           ... t[pos] = t[pos][::-1].upper()
            For example, the following result was generated from a parse
                                                                                                                                          >>> print t
tree of
                                                                                                                                          (S (NP (D EHT) (N GOD)) (VP (V DESAHC) (NP (D EHT)
            the sentence "The announcement astounded us":::
                                                                                                                          (N TAC))))
             \Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
                                                                                                                                       :param order: One of: ``preorder``, ``postorder``, ``bothorder``,
                                                                                                                                           ``leaves``.
                 [.l' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
            See http://www.ling.upenn.edu/advice/latex.html for the
                                                                                                                                  un_chomsky_normal_form(self, expandUnary=True,
                                                                                                                          childChar='|', parentChar='^', unaryChar='+')
LaTeX
            style file for the qtree package.
                                                                                                                                       This method modifies the tree in three ways:
                                                                                                                                         1. Transforms a tree in Chomsky Normal Form back to its
            :return: A latex gtree representation of this tree.
            :rtype: str
                                                                                                                                            original structure (branching greater than two)
                                                                                                                                         2. Removes any parent annotation (if it exists)
                                                                                                                                         3. (optional) expands unary subtrees (if previously
       productions(self)
            Generate the productions that correspond to the non-terminal
                                                                                                                                            collapsed with collapseUnary(...))
nodes of the tree.
            For each subtree of the form (P: C1 C2 ... Cn) this produces a
                                                                                                                                       :param expandUnary: Flag to expand unary or not (default =
production of the
                                                                                                                           True)
```

```
:type expandUnary: bool
                                                                                       used to find node and leaf substrings in "s". By
       :param childChar: A string separating the head node from its
                                                                                       default, both nodes patterns are defined to match any
children in an artificial node (default = "|")
                                                                                       sequence of non-whitespace non-bracket characters.
       :type childChar: str
       :param parentChar: A sting separating the node label from its
                                                                                     :type remove_empty_top_bracketing: bool
parent annotation (default = "^")
                                                                                     :param remove_empty_top_bracketing: If the resulting tree
       :type parentChar: str
                                                                             has
       :param unaryChar: A string joining two non-terminals in a
                                                                                       an empty node label, and is length one, then return its
unary production (default = "+")
                                                                                       single child instead. This is useful for treebank trees,
       :type unaryChar: str
                                                                                       which sometimes contain an extra level of bracketing.
                                                                                     :return: A tree corresponding to the string representation ``s``.
                                                                                       If this class method is called using a subclass of Tree,
    Class methods inherited from Tree:
                                                                                       then it will return a tree of that type.
                                                                                     :rtype: Tree
     convert(cls, tree) from __builtin__.type
       Convert a tree between different subtypes of Tree. "cls"
determines
       which class will be used to encode the new tree.
                                                                                  Data descriptors inherited from Tree:
       :type tree: Tree
                                                                                     dict
       :param tree: The tree that should be converted.
                                                                                     dictionary for instance variables (if defined)
       :return: The new Tree.
                                                                                     weakref
    parse(cls, s, brackets='()', parse_node=None, parse_leaf=None,
                                                                                     list of weak references to the object (if defined)
node_pattern=None, leaf_pattern=None,
remove_empty_top_bracketing=False) from __builtin__.type
       Parse a bracketed tree string and return the resulting tree.
                                                                                  Methods inherited from builtin .list:
       Trees are represented as nested brackettings, such as::
                                                                                    _contains__(...)
        (S (NP (NNP John)) (VP (V runs)))
                                                                                     x.__contains__(y) <==> y in x
       :type s: str
                                                                                    _getattribute__(...)
                                                                                     x. getattribute ('name') <==> x.name
       :param s: The string to parse
       :type brackets: str (length=2)
                                                                                    _iter__(...)
       :param brackets: The bracket characters used to mark the
                                                                                    x.__iter__() <==> iter(x)
          beginning and end of trees and subtrees.
                                                                                    _len__(...)
                                                                                    x.__len__() <==> len(x)
       :type parse_node: function
       :type parse leaf: function
       :param parse_node, parse_leaf: If specified, these functions
                                                                                    reversed__(...)
          are applied to the substrings of "s" corresponding to
                                                                                     L.__reversed__() -- return a reverse iterator over the list
          nodes and leaves (respectively) to obtain the values for
          those nodes and leaves. They should have the following
          signature:
                                                                                    L.__sizeof__() -- size of L in memory, in bytes
           parse node(str) -> value
                                                                                  count(...)
                                                                                     L.count(value) -> integer -- return number of occurrences of
          For example, these functions could be used to parse nodes
                                                                             value
          and leaves whose values should be some type other than
          string (such as ``FeatStruct``).
          Note that by default, node strings and leaf strings are
                                                                                    L.index(value, [start, [stop]]) -> integer -- return first index of
          delimited by whitespace and brackets; to override this
                                                                             value.
          default, use the "node_pattern" and "leaf_pattern"
                                                                                     Raises ValueError if the value is not present.
          arguments.
       :type node pattern: str
                                                                                  Data and other attributes inherited from builtin .list:
       :type leaf_pattern: str
       :param node_pattern, leaf_pattern: Regular expression
                                                                                    _new__ = <built-in method __new__ of type object>
                                                                                    T.__new__(S, ...) -> a new object with type S, a subtype of T
patterns
```

```
class ImmutableProbabilisticTree(ImmutableTree,
nltk.probability.ProbabilisticMixIn)
                                                                                  Data descriptors inherited from ImmutableTree:
    Method resolution order:
       ImmutableProbabilisticTree
                                                                                  node
       ImmutableTree
                                                                                     Get the node value
       Tree
          builtin__.list
       nltk.probability.ProbabilisticMixIn
                                                                                  Methods inherited from Tree:
       builtin .object
                                                                                  add (self, v)
    Methods defined here:
                                                                                  __ge__(self, other)
     __cmp__(self, other)
                                                                                  __getitem__(self, index)
     eq (self, other)
                                                                                  __gt__(self, other)
     __init__(self, node_or_str, children=None, **prob_kwargs)
                                                                                  __le__(self, other)
     __ne__(self, other)
                                                                                  __lt__(self, other)
     __repr__(self)
                                                                                  __mul__(self, v)
     __str__(self)
                                                                                  __radd__(self, v)
    copy(self, deep=False)
                                                                                  rmul (self, v)
                                                                                  chomsky_normal_form(self, factor='right', horzMarkov=None,
     Class methods defined here:
                                                                             vertMarkov=0, childChar='|', parentChar='^')
                                                                                     This method can modify a tree in three ways:
    convert(cls, val) from __builtin__.type
                                                                                      1. Convert a tree into its Chomsky Normal Form (CNF)
                                                                                        equivalent -- Every subtree has either two non-terminals
    Methods inherited from ImmutableTree:
                                                                                        or one terminal as its children. This process requires
     __delitem__(self, index)
                                                                                        the creation of more "artificial" non-terminal nodes.
                                                                                      2. Markov (vertical) smoothing of children in new artificial
     __delslice__(self, i, j)
                                                                                        nodes
                                                                                      3. Horizontal (parent) annotation of nodes
     hash (self)
                                                                                     :param factor: Right or left factoring method (default = "right")
                                                                                     :type factor: str = [left|right]
     __iadd__(self, other)
                                                                                     :param horzMarkov: Markov order for sibling smoothing in
                                                                             artificial nodes (None (default) = include all siblings)
     __imul__(self, other)
                                                                                     :type horzMarkov: int | None
     __setitem__(self, index, value)
                                                                                     :param vertMarkov: Markov order for parent smoothing (0
                                                                             (default) = no vertical annotation)
                                                                                     :type vertMarkov: int | None
     __setslice__(self, i, j, value)
                                                                                     :param childChar: A string used in construction of the artificial
                                                                             nodes, separating the head of the
    append(self, v)
                                                                                                original subtree from the child nodes that have yet
                                                                             to be expanded (default = "|")
     extend(self, v)
                                                                                     :type childChar: str
    pop(self, v=None)
                                                                                     :param parentChar: A string used to separate the node
                                                                             representation from its vertical annotation
                                                                                     :type parentChar: str
    remove(self, v)
                                                                                 collapse_unary(self, collapsePOS=False, collapseRoot=False,
    reverse(self)
                                                                             joinChar='+')
```

sort(self)

Collapse subtrees with a single child (ie. unary productions)

```
into a new non-terminal (Tree node) joined by 'joinChar'.
                                                                                       :rtype: int
       This is useful when working with algorithms that do not allow
       unary productions, and completely removing the unary
                                                                                    leaf treeposition(self, index)
                                                                                       :return: The tree position of the ``index``-th leaf in this
productions
       would require loss of useful information. The Tree is modified
                                                                                          tree. I.e., if "tp=self.leaf treeposition(i)", then
                                                                                          ``self[tp]==self.leaves()[i]``.
       directly (since it is passed by reference) and no value is
returned.
                                                                                       :raise IndexError: If this tree contains fewer than ``index+1``
                                                                                          leaves, or if ``index<0``.
        :param collapsePOS: 'False' (default) will not collapse the
parent of leaf nodes (ie.
                    Part-of-Speech tags) since they are always unary
                                                                                    leaves(self)
productions
                                                                                       Return the leaves of the tree.
        :type collapsePOS: bool
        :param collapseRoot: 'False' (default) will not modify the root
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                               (D the) (N cat))))")
production
                                                                                         >>> t.leaves()
                     if it is unary. For the Penn WSJ treebank corpus,
                                                                                          ['the', 'dog', 'chased', 'the', 'cat']
this corresponds
                     to the TOP -> productions.
        :type collapseRoot: bool
                                                                                       :return: a list containing this tree's leaves.
                                                                                          The order reflects the order of the
       :param joinChar: A string used to connect collapsed node
                                                                                          leaves in the tree's hierarchical structure.
values (default = "+")
       :type joinChar: str
                                                                                       :rtype: list
     draw(self)
                                                                                    pos(self)
       Open a new window containing a graphical diagram of this
                                                                                       Return a sequence of pos-tagged words extracted from the
tree.
                                                                               tree.
     flatten(self)
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
       Return a flat version of the tree, with all non-root non-
                                                                               (D the) (N cat))))")
terminals removed.
                                                                                          >>> t.pos()
                                                                                          [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
(D the) (N cat))))")
                                                                                       :return: a list of tuples containing leaves and pre-terminals
          >>> print t.flatten()
                                                                               (part-of-speech tags).
          (S the dog chased the cat)
                                                                                          The order reflects the order of the leaves in the tree's
                                                                               hierarchical structure.
        :return: a tree consisting of this tree's root connected directly
                                                                                       :rtype: list(tuple)
to
                                                                                    pprint(self, margin=70, indent=0, nodesep=", parens='()',
          its leaves, omitting all intervening non-terminal nodes.
       :rtvpe: Tree
                                                                               quotes=False)
                                                                                       :return: A pretty-printed string representation of this tree.
     freeze(self, leaf_freezer=None)
                                                                                       :param margin: The right margin at which to do line-wrapping.
     height(self)
                                                                                       :type margin: int
       Return the height of the tree.
                                                                                       :param indent: The indentation level at which printing
                                                                                          begins. This number is used to decide how far to indent
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                          subsequent lines.
                                                                                       :type indent: int
(D the) (N cat)))")
          >>> t.height()
                                                                                       :param nodesep: A string that is used to separate the node
                                                                                          from the children. E.g., the default value "':" gives
          >>> print t[0,0]
                                                                                          trees like "(S: (NP: I) (VP: (V: saw) (NP: it)))".
          (D the)
          >>> t[0,0].height()
                                                                                    pprint latex gtree(self)
                                                                                       Returns a representation of the tree compatible with the
                                                                                       LaTeX qtree package. This consists of the string ``\Tree``
                                                                                       followed by the parse tree represented in bracketed notation.
       :return: The height of this tree. The height of a tree
          containing no children is 1; the height of a tree
          containing only leaves is 2; and the height of any other
                                                                                       For example, the following result was generated from a parse
          tree is one plus the maximum of its children's
                                                                               tree of
                                                                                       the sentence "The announcement astounded us"::
          heights.
```

```
\Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
                                                                                                                                     :param order: One of: ``preorder``, ``postorder``, ``bothorder``,
                 [.I' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
                                                                                                                                         ``leaves``.
           See http://www.ling.upenn.edu/advice/latex.html for the
                                                                                                                                un_chomsky_normal_form(self, expandUnary=True,
                                                                                                                         childChar='|', parentChar='^', unaryChar='+')
LaTeX
                                                                                                                                     This method modifies the tree in three ways:
           style file for the qtree package.
                                                                                                                                       1. Transforms a tree in Chomsky Normal Form back to its
           :return: A latex gtree representation of this tree.
                                                                                                                                          original structure (branching greater than two)
           :rtvpe: str
                                                                                                                                       2. Removes any parent annotation (if it exists)
                                                                                                                                       3. (optional) expands unary subtrees (if previously
       productions(self)
                                                                                                                                          collapsed with collapseUnary(...))
           Generate the productions that correspond to the non-terminal
nodes of the tree.
           For each subtree of the form (P: C1 C2 ... Cn) this produces a
                                                                                                                                     :param expandUnary: Flag to expand unary or not (default =
production of the
                                                                                                                         True)
           form P -> C1 C2 ... Cn.
                                                                                                                                     :type expandUnary: bool
                                                                                                                                     :param childChar: A string separating the head node from its
               >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                         children in an artificial node (default = "|")
                                                                                                                                     :type childChar: str
(D the) (N cat))))")
                                                                                                                                     :param parentChar: A sting separating the node label from its
               >>> t.productions()
               [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP,
                                                                                                                         parent annotation (default = "^")
                                                                                                                                     :type parentChar: str
V -> 'chased'.
                                                                                                                                     :param unaryChar: A string joining two non-terminals in a
               NP -> D N, D -> 'the', N -> 'cat']
                                                                                                                         unary production (default = "+")
                                                                                                                                     :type unaryChar: str
           :rtype: list(Production)
       subtrees(self, filter=None)
           Generate all the subtrees of this tree, optionally restricted
                                                                                                                                 Class methods inherited from Tree:
           to trees matching the filter function.
                                                                                                                                parse(cls, s, brackets='()', parse_node=None, parse_leaf=None,
                                                                                                                         node_pattern=None, leaf_pattern=None,
               >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                         remove empty top bracketing=False) from builtin .type
(D the) (N cat)))")
               >>> for s in t.subtrees(lambda t: t.height() == 2):
                                                                                                                                     Parse a bracketed tree string and return the resulting tree.
                       print s
                                                                                                                                     Trees are represented as nested brackettings, such as::
               (D the)
                (N dog)
                                                                                                                                       (S (NP (NNP John)) (VP (V runs)))
                (V chased)
                (D the)
                                                                                                                                     :type s: str
                (N cat)
                                                                                                                                     :param s: The string to parse
            :type filter: function
                                                                                                                                     :type brackets: str (length=2)
                                                                                                                                     :param brackets: The bracket characters used to mark the
            :param filter: the function to filter all local trees
                                                                                                                                         beginning and end of trees and subtrees.
       treeposition_spanning_leaves(self, start, end)
           :return: The tree position of the lowest descendant of this
                                                                                                                                     :type parse_node: function
               tree that dominates "self.leaves()[start:end]".
                                                                                                                                     :type parse leaf: function
            :raise ValueError: if ``end <= start``
                                                                                                                                     :param parse_node, parse_leaf: If specified, these functions
                                                                                                                                         are applied to the substrings of ``s`` corresponding to
                                                                                                                                         nodes and leaves (respectively) to obtain the values for
       treepositions(self, order='preorder')
               >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                                         those nodes and leaves. They should have the following
(D the) (N cat))))")
                                                                                                                                         signature:
               >>> t.treepositions() # doctest: +ELLIPSIS
               [(), (0,), (0, 0), (0, 0, 0), (0, 1), (0, 1, 0), (1,), (1, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (
                                                                                                                                            parse_node(str) -> value
0), ...]
               >>> for pos in t.treepositions('leaves'):
                                                                                                                                         For example, these functions could be used to parse nodes
               ... t[pos] = t[pos][::-1].upper()
                                                                                                                                         and leaves whose values should be some type other than
                                                                                                                                         string (such as ``FeatStruct``).
               >>> print t
               (S (NP (D EHT) (N GOD)) (VP (V DESAHC) (NP (D EHT)
                                                                                                                                         Note that by default, node strings and leaf strings are
```

(N TAC))))

delimited by whitespace and brackets; to override this

```
default, use the "node_pattern" and "leaf_pattern"
                                                                                      L.count(value) -> integer -- return number of occurrences of
          arguments.
                                                                              value
       :type node_pattern: str
                                                                                   index(...)
       :type leaf_pattern: str
                                                                                      L.index(value, [start, [stop]]) -> integer -- return first index of
       :param node_pattern, leaf_pattern: Regular expression
                                                                              value.
patterns
                                                                                      Raises ValueError if the value is not present.
          used to find node and leaf substrings in "s". By
          default, both nodes patterns are defined to match any
                                                                                   insert(...)
          sequence of non-whitespace non-bracket characters.
                                                                                      L.insert(index, object) -- insert object before index
        :type remove_empty_top_bracketing: bool
        :param remove_empty_top_bracketing: If the resulting tree
                                                                                   Data and other attributes inherited from builtin .list:
has
                                                                                     _new__ = <built-in method __new__ of type object>
          an empty node label, and is length one, then return its
          single child instead. This is useful for treebank trees,
                                                                                      T. new (S, ...) -> a new object with type S, a subtype of T
          which sometimes contain an extra level of bracketing.
       :return: A tree corresponding to the string representation ``s``.
                                                                                   Methods inherited from nltk.probability.ProbabilisticMixIn:
          If this class method is called using a subclass of Tree.
          then it will return a tree of that type.
                                                                                   logprob(self)
       :rtype: Tree
                                                                                      Return ``log(p)``, where ``p`` is the probability associated
                                                                                      with this object.
     Data descriptors inherited from Tree:
                                                                                      :rtype: float
      dict
                                                                                   prob(self)
                                                                                      Return the probability associated with this object.
       dictionary for instance variables (if defined)
       weakref
                                                                                      :rtype: float
       list of weak references to the object (if defined)
                                                                                   set_logprob(self, logprob)
                                                                                      Set the log probability associated with this object to
     Methods inherited from builtin .list:
                                                                                      "logprob". I.e., set the probability associated with this
                                                                                      object to ``2**(logprob)``.
      _contains__(...)
       x.__contains__(y) <==> y in x
                                                                                      :param logprob: The new log probability
                                                                                      :type logprob: float
      __getattribute__(...)
       x. getattribute ('name') <==> x.name
                                                                                   set prob(self, prob)
                                                                                      Set the probability associated with this object to "prob".
      __getslice__(...)
       x.__getslice__(i, j) <==> x[i:j]
                                                                                      :param prob: The new probability
                                                                                      :type prob: float
       Use of negative indices is not supported.
                                                                                 class ImmutableTree(Tree)
                                                                                   Method resolution order:
      __iter__(...)
       x.__iter__() <==> iter(x)
                                                                                      ImmutableTree
                                                                                      Tree
                                                                                      __builtin__.list
     __len__(...)
       x. len () <==> len(x)
                                                                                      __builtin__.object
                                                                                   Methods defined here:
       reversed (...)
       L.__reversed__() -- return a reverse iterator over the list
                                                                                   __delitem__(self, index)
      sizeof (...)
       L. sizeof () -- size of L in memory, in bytes
                                                                                    delslice (self, i, j)
    count(...)
                                                                                    __hash__(self)
```

```
__iadd__(self, other)
                                                                                      1. Convert a tree into its Chomsky Normal Form (CNF)
     imul (self, other)
                                                                                        equivalent -- Every subtree has either two non-terminals
                                                                                        or one terminal as its children. This process requires
    __init__(self, node_or_str, children=None)
                                                                                        the creation of more "artificial" non-terminal nodes.
                                                                                      2. Markov (vertical) smoothing of children in new artificial
     __setitem__(self, index, value)
                                                                                        nodes
                                                                                      3. Horizontal (parent) annotation of nodes
     setslice (self, i, j, value)
                                                                                     :param factor: Right or left factoring method (default = "right")
    append(self, v)
                                                                                     :type factor: str = [left|right]
                                                                                     :param horzMarkov: Markov order for sibling smoothing in
    extend(self, v)
                                                                             artificial nodes (None (default) = include all siblings)
                                                                                     :type horzMarkov: int | None
                                                                                     :param vertMarkov: Markov order for parent smoothing (0
    pop(self, v=None)
                                                                             (default) = no vertical annotation)
                                                                                     :type vertMarkov: int | None
    remove(self, v)
                                                                                     :param childChar: A string used in construction of the artificial
                                                                             nodes, separating the head of the
    reverse(self)
                                                                                                original subtree from the child nodes that have yet
    sort(self)
                                                                             to be expanded (default = "|")
                                                                                     :type childChar: str
                                                                                     :param parentChar: A string used to separate the node
    Data descriptors defined here:
                                                                             representation from its vertical annotation
                                                                                     :type parentChar: str
    node
       Get the node value
                                                                                 collapse unary(self, collapsePOS=False, collapseRoot=False,
                                                                             joinChar='+')
                                                                                     Collapse subtrees with a single child (ie. unary productions)
    Methods inherited from Tree:
                                                                                     into a new non-terminal (Tree node) joined by 'joinChar'.
                                                                                     This is useful when working with algorithms that do not allow
     __add__(self, v)
                                                                                     unary productions, and completely removing the unary
                                                                             productions
                                                                                     would require loss of useful information. The Tree is modified
     __eq__(self, other)
                                                                                     directly (since it is passed by reference) and no value is
                                                                             returned.
     __ge__(self, other)
     __getitem__(self, index)
                                                                                     :param collapsePOS: 'False' (default) will not collapse the
                                                                             parent of leaf nodes (ie.
     gt (self, other)
                                                                                                  Part-of-Speech tags) since they are always unary
                                                                             productions
                                                                                     :type collapsePOS: bool
     __le__(self, other)
                                                                                     :param collapseRoot: 'False' (default) will not modify the root
                                                                             production
     __lt__(self, other)
                                                                                                  if it is unary. For the Penn WSJ treebank corpus,
    __mul__(self, v)
                                                                             this corresponds
                                                                                                  to the TOP -> productions.
     __ne__(self, other)
                                                                                     :type collapseRoot: bool
                                                                                     :param joinChar: A string used to connect collapsed node
                                                                             values (default = "+")
     __radd__(self, v)
                                                                                     :type joinChar: str
    __repr__(self)
                                                                                  copy(self, deep=False)
     __rmul__(self, v)
                                                                                  draw(self)
     __str__(self)
                                                                                     Open a new window containing a graphical diagram of this
                                                                             tree.
    chomsky_normal_form(self, factor='right', horzMarkov=None,
vertMarkov=0, childChar='|', parentChar='^')
                                                                                 | flatten(self)
       This method can modify a tree in three ways:
```

```
Return a flat version of the tree, with all non-root non-
                                                                                            >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
terminals removed.
                                                                                 (D the) (N cat)))")
                                                                                            >>> t.pos()
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                            [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
(D the) (N cat))))")
          >>> print t.flatten()
                                                                                         :return: a list of tuples containing leaves and pre-terminals
          (S the dog chased the cat)
                                                                                 (part-of-speech tags).
                                                                                            The order reflects the order of the leaves in the tree's
        :return: a tree consisting of this tree's root connected directly
                                                                                 hierarchical structure.
to
                                                                                         :rtype: list(tuple)
          its leaves, omitting all intervening non-terminal nodes.
        :rtype: Tree
                                                                                    pprint(self, margin=70, indent=0, nodesep=", parens='()',
                                                                                 quotes=False)
    freeze(self, leaf_freezer=None)
                                                                                         :return: A pretty-printed string representation of this tree.
                                                                                         :rtype: str
    height(self)
                                                                                         :param margin: The right margin at which to do line-wrapping.
       Return the height of the tree.
                                                                                         :type margin: int
                                                                                         :param indent: The indentation level at which printing
                                                                                            begins. This number is used to decide how far to indent
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                            subsequent lines.
(D the) (N cat))))")
          >>> t.height()
                                                                                         :type indent: int
                                                                                         :param nodesep: A string that is used to separate the node
                                                                                            from the children. E.g., the default value ``': '`` gives
          >>> print t[0,0]
                                                                                            trees like ``(S: (NP: I) (VP: (V: saw) (NP: it)))``.
          (D the)
          >>> t[0,0].height()
                                                                                      pprint latex gtree(self)
                                                                                         Returns a representation of the tree compatible with the
                                                                                         LaTeX gtree package. This consists of the string "\Tree"
       :return: The height of this tree. The height of a tree
          containing no children is 1; the height of a tree
                                                                                         followed by the parse tree represented in bracketed notation.
          containing only leaves is 2; and the height of any other
          tree is one plus the maximum of its children's
                                                                                         For example, the following result was generated from a parse
          heights.
                                                                                 tree of
       :rtype: int
                                                                                         the sentence "The announcement astounded us"::
     leaf_treeposition(self, index)
                                                                                          \Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
       :return: The tree position of the ``index``-th leaf in this
                                                                                             [.I' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
          tree. I.e., if ``tp=self.leaf_treeposition(i)``, then
           ``self[tp]==self.leaves()[i]``.
                                                                                         See http://www.ling.upenn.edu/advice/latex.html for the
                                                                                 LaTeX
       :raise IndexError: If this tree contains fewer than ``index+1``
                                                                                         style file for the gtree package.
          leaves, or if ``index<0``.
                                                                                         :return: A latex gtree representation of this tree.
     leaves(self)
                                                                                         :rtvpe: str
       Return the leaves of the tree.
                                                                                      productions(self)
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                         Generate the productions that correspond to the non-terminal
                                                                                 nodes of the tree.
(D the) (N cat))))")
          >>> t.leaves()
                                                                                         For each subtree of the form (P: C1 C2 ... Cn) this produces a
                                                                                 production of the
          ['the', 'dog', 'chased', 'the', 'cat']
                                                                                         form P -> C1 C2 ... Cn.
       :return: a list containing this tree's leaves.
          The order reflects the order of the
                                                                                            >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
          leaves in the tree's hierarchical structure.
                                                                                 (D the) (N cat))))")
       :rtype: list
                                                                                            >>> t.productions()
                                                                                            [S \rightarrow NP VP, NP \rightarrow D N, D \rightarrow 'the', N \rightarrow 'dog', VP \rightarrow V NP,
                                                                                 V -> 'chased',
       Return a sequence of pos-tagged words extracted from the
                                                                                            NP -> D N, D -> 'the', N -> 'cat']
tree.
                                                                                         :rtype: list(Production)
```

```
subtrees(self, filter=None)
       Generate all the subtrees of this tree, optionally restricted
                                                                                  Class methods inherited from Tree:
       to trees matching the filter function.
                                                                                  convert(cls, tree) from __builtin__.type
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                     Convert a tree between different subtypes of Tree. "cls"
(D the) (N cat))))")
                                                                             determines
          >>> for s in t.subtrees(lambda t: t.height() == 2):
                                                                                     which class will be used to encode the new tree.
              print s
                                                                                     :type tree: Tree
          (D the)
                                                                                     :param tree: The tree that should be converted.
          (N dog)
          (V chased)
                                                                                     :return: The new Tree.
          (D the)
          (N cat)
                                                                                  parse(cls, s, brackets='()', parse_node=None, parse_leaf=None,
                                                                             node_pattern=None, leaf_pattern=None,
       :type filter: function
                                                                             remove empty top bracketing=False) from builtin .type
       :param filter: the function to filter all local trees
                                                                                     Parse a bracketed tree string and return the resulting tree.
                                                                                     Trees are represented as nested brackettings, such as::
    treeposition_spanning_leaves(self, start, end)
       :return: The tree position of the lowest descendant of this
                                                                                      (S (NP (NNP John)) (VP (V runs)))
          tree that dominates "self.leaves()[start:end]".
       :raise ValueError: if ``end <= start``
                                                                                     :type s: str
                                                                                     :param s: The string to parse
    treepositions(self, order='preorder')
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                     :type brackets: str (length=2)
(D the) (N cat))))")
                                                                                     :param brackets: The bracket characters used to mark the
                                                                                       beginning and end of trees and subtrees.
          >>> t.treepositions() # doctest: +ELLIPSIS
          [(), (0,), (0, 0), (0, 0, 0), (0, 1), (0, 1, 0), (1, 0), (1, 0), (1, 0, 0)]
0), ...]
                                                                                     :type parse node: function
                                                                                     :type parse_leaf: function
          >>> for pos in t.treepositions('leaves'):
          ... t[pos] = t[pos][::-1].upper()
                                                                                     :param parse_node, parse_leaf: If specified, these functions
         >>> print t
                                                                                       are applied to the substrings of "s" corresponding to
                                                                                       nodes and leaves (respectively) to obtain the values for
          (S (NP (D EHT) (N GOD)) (VP (V DESAHC) (NP (D EHT)
(N TAC))))
                                                                                       those nodes and leaves. They should have the following
                                                                                       signature:
       :param order: One of: ``preorder``, ``postorder``, ``bothorder``,
          `leaves``.
                                                                                         parse_node(str) -> value
    un_chomsky_normal_form(self, expandUnary=True,
                                                                                       For example, these functions could be used to parse nodes
childChar='|', parentChar='^', unaryChar='+')
                                                                                       and leaves whose values should be some type other than
       This method modifies the tree in three ways:
                                                                                       string (such as ``FeatStruct``).
                                                                                       Note that by default, node strings and leaf strings are
        1. Transforms a tree in Chomsky Normal Form back to its
                                                                                       delimited by whitespace and brackets; to override this
          original structure (branching greater than two)
                                                                                       default, use the "node pattern" and "leaf pattern"
        2. Removes any parent annotation (if it exists)
                                                                                       arguments.
        3. (optional) expands unary subtrees (if previously
          collapsed with collapseUnary(...))
                                                                                     :type node_pattern: str
                                                                                     :type leaf pattern: str
       :param expandUnary: Flag to expand unary or not (default =
                                                                                     :param node_pattern, leaf_pattern: Regular expression
True)
                                                                             patterns
                                                                                       used to find node and leaf substrings in "s". By
       :type expandUnary: bool
       :param childChar: A string separating the head node from its
                                                                                       default, both nodes patterns are defined to match any
children in an artificial node (default = "|")
                                                                                       sequence of non-whitespace non-bracket characters.
       :type childChar: str
       :param parentChar: A sting separating the node label from its
                                                                                     :type remove_empty_top_bracketing: bool
parent annotation (default = "^")
                                                                                     :param remove_empty_top_bracketing: If the resulting tree
       :type parentChar: str
                                                                             has
       :param unaryChar: A string joining two non-terminals in a
                                                                                       an empty node label, and is length one, then return its
unary production (default = "+")
                                                                                       single child instead. This is useful for treebank trees,
       :type unaryChar: str
                                                                                       which sometimes contain an extra level of bracketing.
```

```
:return: A tree corresponding to the string representation "s".
                                                                                  multi-parented trees. The following are methods for querying
          If this class method is called using a subclass of Tree,
                                                                               the
          then it will return a tree of that type.
                                                                                  | structure of a multi-parented tree: "parents()",
                                                                                `parent_indices()``,
       :rtype: Tree
                                                                                    ``left_siblings()``, ``right_siblings()``, ``roots``, ``treepositions``.
     Data descriptors inherited from Tree:
                                                                                    Each "MultiParentedTree" may have zero or more parents. In
                                                                                    particular, subtrees may be shared. If a single
                                                                                     ``MultiParentedTree`` is used as multiple children of the same
       dict
                                                                                    parent, then that parent will appear multiple times in its
       dictionary for instance variables (if defined)
                                                                                     `parents()`` method.
      weakref
       list of weak references to the object (if defined)
                                                                                    "MultiParentedTrees" should never be used in the same tree
                                                                               as
                                                                                  | ``Trees`` or ``ParentedTrees``. Mixing tree implementations
     Methods inherited from builtin .list:
                                                                                  result in incorrect parent pointers and in "TypeError"
                                                                               exceptions.
       _contains__(...)
       x_contains_(y) <==> y in x
                                                                                    Method resolution order:
       _getattribute__(...)
                                                                                       MultiParentedTree
       x.__getattribute__('name') <==> x.name
                                                                                       AbstractParentedTree
                                                                                       Tree
                                                                                         _builtin___.list
       _getslice__(...)
       x.__getslice__(i, j) <==> x[i:j]
                                                                                       __builtin__.object
       Use of negative indices is not supported.
                                                                                    Methods defined here:
                                                                                     __init__(self, node_or_str, children=None)
       x.__iter__() <==> iter(x)
                                                                                    left_siblings(self)
                                                                                       A list of all left siblings of this tree, in any of its parent
     __len__(...)
       x. len () <==> len(x)
                                                                                       trees. A tree may be its own left sibling if it is used as
                                                                                       multiple contiguous children of the same parent. A tree may
     __reversed__(...)
                                                                                       appear multiple times in this list if it is the left sibling
       L.__reversed__() -- return a reverse iterator over the list
                                                                                       of this tree with respect to multiple parents.
       _sizeof___(...)
                                                                                       :type: list(MultiParentedTree)
       L.__sizeof__() -- size of L in memory, in bytes
                                                                                    parent indices(self, parent)
                                                                                       Return a list of the indices where this tree occurs as a child
       L.count(value) -> integer -- return number of occurrences of
                                                                                       of "parent". If this child does not occur as a child of
                                                                                       "parent", then the empty list is returned. The following is
value
                                                                                       always true::
    index(...)
       L.index(value, [start, [stop]]) -> integer -- return first index of
                                                                                        for parent_index in ptree.parent_indices(parent):
                                                                                           parent[parent index] is ptree
value.
       Raises ValueError if the value is not present.
                                                                                    parents(self)
                                                                                       The set of parents of this tree. If this tree has no parents,
    insert(...)
                                                                                       then "parents" is the empty set. To check if a tree is used
       L.insert(index, object) -- insert object before index
                                                                                       as multiple children of the same parent, use the
                                                                                       "parent indices()" method.
     Data and other attributes inherited from builtin .list:
                                                                                       :type: list(MultiParentedTree)
       _new__ = <built-in method __new__ of type object>
       T. new (S, ...) -> a new object with type S, a subtype of T
                                                                                    right siblings(self)
                                                                                       A list of all right siblings of this tree, in any of its parent
  class MultiParentedTree(AbstractParentedTree)
                                                                                       trees. A tree may be its own right sibling if it is used as
```

multiple contiguous children of the same parent. A tree may

A "Tree" that automatically maintains parent pointers for

```
appear multiple times in this list if it is the right sibling
                                                                               __lt__(self, other)
  of this tree with respect to multiple parents.
                                                                               mul (self, v)
  :type: list(MultiParentedTree)
                                                                              __ne__(self, other)
roots(self)
  The set of all roots of this tree. This set is formed by
                                                                               __radd__(self, v)
  tracing all possible parent paths until trees with no parents
  are found.
                                                                               __repr__(self)
  :type: list(MultiParentedTree)
                                                                               __rmul__(self, v)
treepositions(self, root)
                                                                               __str__(self)
  Return a list of all tree positions that can be used to reach
  this multi-parented tree starting from "root". I.e., the
                                                                              chomsky normal form(self, factor='right', horzMarkov=None,
                                                                         vertMarkov=0, childChar='|', parentChar='^')
  following is always true::
                                                                                 This method can modify a tree in three ways:
   for treepos in ptree.treepositions(root):
      root[treepos] is ptree
                                                                                   1. Convert a tree into its Chomsky Normal Form (CNF)
                                                                                     equivalent -- Every subtree has either two non-terminals
                                                                                    or one terminal as its children. This process requires
Methods inherited from AbstractParentedTree:
                                                                                     the creation of more "artificial" non-terminal nodes.
                                                                                   2. Markov (vertical) smoothing of children in new artificial
 delitem (self, index)
                                                                                   3. Horizontal (parent) annotation of nodes
__delslice__(self, start, stop)
                                                                                 :param factor: Right or left factoring method (default = "right")
                                                                                 :type factor: str = [left|right]
__getslice__(self, start, stop)
                                                                                 :param horzMarkov: Markov order for sibling smoothing in
__setitem__(self, index, value)
                                                                         artificial nodes (None (default) = include all siblings)
                                                                                 :type horzMarkov: int | None
                                                                                 :param vertMarkov: Markov order for parent smoothing (0
__setslice__(self, start, stop, value)
                                                                         (default) = no vertical annotation)
                                                                                 :type vertMarkov: int | None
append(self, child)
                                                                                 :param childChar: A string used in construction of the artificial
extend(self, children)
                                                                         nodes, separating the head of the
                                                                                             original subtree from the child nodes that have yet
                                                                         to be expanded (default = "|")
insert(self, index, child)
                                                                                 :type childChar: str
                                                                                 :param parentChar: A string used to separate the node
pop(self, index=-1)
                                                                         representation from its vertical annotation
remove(self, child)
                                                                                 :type parentChar: str
  # n.b.: like `list`, this is done by equality, not identity!
  # To remove a specific child, use del ptree[i].
                                                                              collapse_unary(self, collapsePOS=False, collapseRoot=False,
                                                                         joinChar='+')
                                                                                 Collapse subtrees with a single child (ie. unary productions)
                                                                                 into a new non-terminal (Tree node) joined by 'joinChar'.
Methods inherited from Tree:
                                                                                 This is useful when working with algorithms that do not allow
__add__(self, v)
                                                                                 unary productions, and completely removing the unary
                                                                         productions
__eq__(self, other)
                                                                                 would require loss of useful information. The Tree is modified
                                                                                 directly (since it is passed by reference) and no value is
                                                                         returned.
ge (self, other)
__getitem__(self, index)
                                                                                 :param collapsePOS: 'False' (default) will not collapse the
                                                                         parent of leaf nodes (ie.
                                                                                              Part-of-Speech tags) since they are always unary
__gt__(self, other)
                                                                         productions
__le__(self, other)
                                                                                 :type collapsePOS: bool
```

```
:param collapseRoot: 'False' (default) will not modify the root
                                                                                        Return the leaves of the tree.
production
                     if it is unary. For the Penn WSJ treebank corpus,
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
this corresponds
                                                                                (D the) (N cat))))")
                     to the TOP -> productions.
                                                                                          >>> t.leaves()
                                                                                          ['the', 'dog', 'chased', 'the', 'cat']
        :type collapseRoot: bool
        :param joinChar: A string used to connect collapsed node
values (default = "+")
                                                                                        :return: a list containing this tree's leaves.
       :type joinChar: str
                                                                                          The order reflects the order of the
                                                                                          leaves in the tree's hierarchical structure.
     copy(self, deep=False)
                                                                                        :rtype: list
     draw(self)
                                                                                     pos(self)
       Open a new window containing a graphical diagram of this
                                                                                        Return a sequence of pos-tagged words extracted from the
tree.
                                                                                tree.
    flatten(self)
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
       Return a flat version of the tree, with all non-root non-
                                                                                (D the) (N cat))))")
terminals removed.
                                                                                          >>> t.pos()
                                                                                          [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                        :return: a list of tuples containing leaves and pre-terminals
(D the) (N cat))))")
          >>> print t.flatten()
                                                                                (part-of-speech tags).
          (S the dog chased the cat)
                                                                                          The order reflects the order of the leaves in the tree's
                                                                                hierarchical structure.
        :return: a tree consisting of this tree's root connected directly
                                                                                        :rtype: list(tuple)
to
          its leaves, omitting all intervening non-terminal nodes.
                                                                                    | pprint(self, margin=70, indent=0, nodesep=", parens='()',
        :rtype: Tree
                                                                                quotes=False)
                                                                                        :return: A pretty-printed string representation of this tree.
    freeze(self, leaf_freezer=None)
                                                                                        :rtvpe: str
                                                                                        :param margin: The right margin at which to do line-wrapping.
     height(self)
                                                                                        :type margin: int
                                                                                        :param indent: The indentation level at which printing
       Return the height of the tree.
                                                                                           begins. This number is used to decide how far to indent
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                          subsequent lines.
                                                                                        :type indent: int
(D the) (N cat))))")
                                                                                        :param nodesep: A string that is used to separate the node
          >>> t.height()
                                                                                          from the children. E.g., the default value "':" gives
                                                                                          trees like ``(S: (NP: I) (VP: (V: saw) (NP: it)))``.
          >>> print t[0,0]
          (D the)
          >>> t[0,0].height()
                                                                                     pprint_latex_qtree(self)
                                                                                        Returns a representation of the tree compatible with the
                                                                                        LaTeX gtree package. This consists of the string "\Tree"
       :return: The height of this tree. The height of a tree
                                                                                        followed by the parse tree represented in bracketed notation.
          containing no children is 1; the height of a tree
          containing only leaves is 2; and the height of any other
                                                                                        For example, the following result was generated from a parse
          tree is one plus the maximum of its children's
                                                                                tree of
          heights.
                                                                                        the sentence "The announcement astounded us":::
       :rtype: int
                                                                                         \Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
                                                                                            [.l' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
     leaf treeposition(self, index)
       :return: The tree position of the ``index``-th leaf in this
          tree. I.e., if "tp=self.leaf_treeposition(i)", then
                                                                                        See http://www.ling.upenn.edu/advice/latex.html for the
           ``self[tp]==self.leaves()[i]``.
                                                                                LaTeX
                                                                                        style file for the qtree package.
       :raise IndexError: If this tree contains fewer than ``index+1``
          leaves, or if ``index<0``.
                                                                                        :return: A latex qtree representation of this tree.
                                                                                        :rtype: str
```

leaves(self)

```
productions(self)
       Generate the productions that correspond to the non-terminal
                                                                                  Class methods inherited from Tree:
nodes of the tree.
       For each subtree of the form (P: C1 C2 ... Cn) this produces a
production of the
                                                                                  convert(cls, tree) from __builtin__.type
       form P -> C1 C2 ... Cn.
                                                                                    Convert a tree between different subtypes of Tree. "cls"
                                                                             determines
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                    which class will be used to encode the new tree.
(D the) (N cat))))")
          >>> t.productions()
                                                                                    :type tree: Tree
          [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP,
                                                                                     :param tree: The tree that should be converted.
                                                                                     :return: The new Tree.
V -> 'chased',
          NP -> D N, D -> 'the', N -> 'cat']
                                                                                  parse(cls, s, brackets='()', parse_node=None, parse_leaf=None,
       :rtype: list(Production)
                                                                             node pattern=None, leaf pattern=None,
                                                                             remove empty top bracketing=False) from builtin .type
                                                                                    Parse a bracketed tree string and return the resulting tree.
    subtrees(self, filter=None)
                                                                                    Trees are represented as nested brackettings, such as::
       Generate all the subtrees of this tree, optionally restricted
       to trees matching the filter function.
                                                                                      (S (NP (NNP John)) (VP (V runs)))
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
(D the) (N cat))))")
                                                                                    :type s: str
          >>> for s in t.subtrees(lambda t: t.height() == 2):
                                                                                     :param s: The string to parse
              print s
          (D the)
                                                                                    :type brackets: str (length=2)
          (N dog)
                                                                                     :param brackets: The bracket characters used to mark the
          (V chased)
                                                                                       beginning and end of trees and subtrees.
          (D the)
          (N cat)
                                                                                     :type parse_node: function
                                                                                     :type parse_leaf: function
                                                                                     :param parse_node, parse_leaf: If specified, these functions
       :type filter: function
                                                                                       are applied to the substrings of ``s`` corresponding to
       :param filter: the function to filter all local trees
                                                                                       nodes and leaves (respectively) to obtain the values for
                                                                                       those nodes and leaves. They should have the following
    treeposition spanning leaves(self, start, end)
       :return: The tree position of the lowest descendant of this
                                                                                       signature:
          tree that dominates "self.leaves()[start:end]".
       :raise ValueError: if ``end <= start``
                                                                                         parse_node(str) -> value
    un_chomsky_normal_form(self, expandUnary=True,
                                                                                       For example, these functions could be used to parse nodes
childChar='|', parentChar='^', unaryChar='+')
                                                                                       and leaves whose values should be some type other than
       This method modifies the tree in three ways:
                                                                                       string (such as "FeatStruct").
                                                                                       Note that by default, node strings and leaf strings are
                                                                                       delimited by whitespace and brackets; to override this
        1. Transforms a tree in Chomsky Normal Form back to its
          original structure (branching greater than two)
                                                                                       default, use the "node_pattern" and "leaf_pattern"
        2. Removes any parent annotation (if it exists)
                                                                                       arguments.
        3. (optional) expands unary subtrees (if previously
          collapsed with collapseUnary(...))
                                                                                     :type node pattern: str
                                                                                     :type leaf_pattern: str
       :param expandUnary: Flag to expand unary or not (default =
                                                                                     :param node_pattern, leaf_pattern: Regular expression
True)
                                                                             patterns
       :type expandUnary: bool
                                                                                       used to find node and leaf substrings in "s". By
       :param childChar: A string separating the head node from its
                                                                                       default, both nodes patterns are defined to match any
children in an artificial node (default = "|")
                                                                                       sequence of non-whitespace non-bracket characters.
       :type childChar: str
       :param parentChar: A sting separating the node label from its
                                                                                     :type remove_empty_top_bracketing: bool
parent annotation (default = "^")
                                                                                    :param remove_empty_top_bracketing: If the resulting tree
       :type parentChar: str
                                                                             has
       :param unaryChar: A string joining two non-terminals in a
                                                                                       an empty node label, and is length one, then return its
unary production (default = "+")
                                                                                       single child instead. This is useful for treebank trees,
       :type unaryChar: str
                                                                                       which sometimes contain an extra level of bracketing.
```

```
Data and other attributes inherited from __builtin__.list:
       :return: A tree corresponding to the string representation "s".
          If this class method is called using a subclass of Tree,
                                                                                   hash = None
          then it will return a tree of that type.
       :rtype: Tree
                                                                                   __new__ = <built-in method __new__ of type object>
                                                                                      T. new (S, ...) -> a new object with type S, a subtype of T
     Data descriptors inherited from Tree:
                                                                                 class ParentedTree(AbstractParentedTree)
                                                                                   A "Tree" that automatically maintains parent pointers for
                                                                                   single-parented trees. The following are methods for querying
       dict
                                                                                   the structure of a parented tree: "parent", "parent_index",
       dictionary for instance variables (if defined)
                                                                                   ``left_sibling``, ``right_sibling``, ``root``, ``treeposition``.
       weakref
       list of weak references to the object (if defined)
                                                                                   Each "ParentedTree" may have at most one parent. In
                                                                                   particular, subtrees may not be shared. Any attempt to reuse a
                                                                                   single "ParentedTree" as a child of more than one parent (or
                                                                                   as multiple children of the same parent) will cause a
     Methods inherited from builtin .list:
                                                                                   "ValueError" exception to be raised.
      _contains__(...)
       x.__contains__(y) <==> y in x
                                                                                   "ParentedTrees" should never be used in the same tree as
                                                                              "Trees"
                                                                                 or ``MultiParentedTrees``. Mixing tree implementations may
     __getattribute__(...)
       x.__getattribute__('name') <==> x.name
                                                                              result
                                                                                   in incorrect parent pointers and in "TypeError" exceptions.
      _iadd__(...)
       x._iadd_(y) <==> x+=y
                                                                                   Method resolution order:
                                                                                      ParentedTree
                                                                                      AbstractParentedTree
     ___imul___(...)
       x.__imul__(y) <==> x*=y
                                                                                      Tree
                                                                                      __builtin__.list
                                                                                      __builtin__.object
      __iter__(...)
       x.__iter__() <==> iter(x)
                                                                                   Methods defined here:
      len (...)
       x.__len__() <==> len(x)
                                                                                   __init__(self, node_or_str, children=None)
     __reversed__(...)
                                                                                   left sibling(self)
       L. reversed__() -- return a reverse iterator over the list
                                                                                      The left sibling of this tree, or None if it has none.
      sizeof (...)
                                                                                   parent(self)
       L. sizeof () -- size of L in memory, in bytes
                                                                                      The parent of this tree, or None if it has no parent.
     count(...)
                                                                                   parent index(self)
       L.count(value) -> integer -- return number of occurrences of
                                                                                      The index of this tree in its parent. I.e.,
                                                                                      ``ptree.parent()[ptree.parent_index()] is ptree``. Note that
value
                                                                                      "ptree.parent index()" is not necessarily equal to
                                                                                      ``ptree.parent.index(ptree)``, since the ``index()`` method
    index(...)
                                                                                      returns the first child that is equal to its argument.
       L.index(value, [start, [stop]]) -> integer -- return first index of
value.
       Raises ValueError if the value is not present.
                                                                                   right_sibling(self)
                                                                                      The right sibling of this tree, or None if it has none.
    reverse(...)
       L.reverse() -- reverse *IN PLACE*
                                                                                   root(self)
                                                                                      The root of this tree. I.e., the unique ancestor of this tree
                                                                                      whose parent is None. If "ptree.parent()" is None, then
    sort(...)
       L.sort(cmp=None, key=None, reverse=False) -- stable sort
                                                                                      "ptree" is its own root.
*IN PLACE*:
       cmp(x, y) \rightarrow -1, 0, 1
                                                                                   treeposition(self)
                                                                                      The tree position of this tree, relative to the root of the
                                                                                      tree. I.e., "ptree.root[ptree.treeposition] is ptree".
```

```
equivalent -- Every subtree has either two non-terminals
                                                                                         or one terminal as its children. This process requires
    Methods inherited from AbstractParentedTree:
                                                                                         the creation of more "artificial" non-terminal nodes.
                                                                                       2. Markov (vertical) smoothing of children in new artificial
     __delitem__(self, index)
                                                                                         nodes
                                                                                       3. Horizontal (parent) annotation of nodes
      _delslice__(self, start, stop)
                                                                                      :param factor: Right or left factoring method (default = "right")
     getslice (self, start, stop)
                                                                                      :type factor: str = [left|right]
                                                                                      :param horzMarkov: Markov order for sibling smoothing in
                                                                              artificial nodes (None (default) = include all siblings)
     __setitem__(self, index, value)
                                                                                      :type horzMarkov: int | None
     __setslice__(self, start, stop, value)
                                                                                      :param vertMarkov: Markov order for parent smoothing (0
                                                                              (default) = no vertical annotation)
                                                                                      :type vertMarkov: int | None
    append(self, child)
                                                                                      :param childChar: A string used in construction of the artificial
                                                                              nodes, separating the head of the
    extend(self, children)
                                                                                                 original subtree from the child nodes that have yet
                                                                              to be expanded (default = "|")
    insert(self, index, child)
                                                                                      :type childChar: str
    pop(self, index=-1)
                                                                                      :param parentChar: A string used to separate the node
                                                                              representation from its vertical annotation
    remove(self, child)
                                                                                      :type parentChar: str
       # n.b.: like `list`, this is done by equality, not identity!
       # To remove a specific child, use del ptree[i].
                                                                                 collapse_unary(self, collapsePOS=False, collapseRoot=False,
                                                                              ioinChar='+')
                                                                                      Collapse subtrees with a single child (ie. unary productions)
    Methods inherited from Tree:
                                                                                      into a new non-terminal (Tree node) joined by 'joinChar'.
                                                                                      This is useful when working with algorithms that do not allow
                                                                                      unary productions, and completely removing the unary
     __add__(self, v)
                                                                              productions
                                                                                     would require loss of useful information. The Tree is modified
     __eq__(self, other)
                                                                                      directly (since it is passed by reference) and no value is
                                                                              returned.
     __ge__(self, other)
                                                                                      :param collapsePOS: 'False' (default) will not collapse the
     __getitem__(self, index)
                                                                              parent of leaf nodes (ie.
     __gt__(self, other)
                                                                                                  Part-of-Speech tags) since they are always unary
                                                                              productions
                                                                                      :type collapsePOS: bool
     le (self, other)
                                                                                      :param collapseRoot: 'False' (default) will not modify the root
     __lt__(self, other)
                                                                              production
                                                                                                   if it is unary. For the Penn WSJ treebank corpus,
                                                                              this corresponds
     __mul__(self, v)
                                                                                                   to the TOP -> productions.
     __ne__(self, other)
                                                                                      :type collapseRoot: bool
                                                                                      :param joinChar: A string used to connect collapsed node
                                                                              values (default = "+")
     __radd__(self, v)
                                                                                      :type joinChar: str
     __repr__(self)
                                                                                   copy(self, deep=False)
     __rmul__(self, v)
                                                                                   draw(self)
                                                                                      Open a new window containing a graphical diagram of this
     __str__(self)
                                                                              tree.
    chomsky_normal_form(self, factor='right', horzMarkov=None,
vertMarkov=0, childChar='|', parentChar='^')
                                                                                   flatten(self)
       This method can modify a tree in three ways:
                                                                                      Return a flat version of the tree, with all non-root non-
                                                                              terminals removed.
         1. Convert a tree into its Chomsky Normal Form (CNF)
```

```
>>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
(D the) (N cat))))")
                                                                                        :return: a list of tuples containing leaves and pre-terminals
          >>> print t.flatten()
                                                                                (part-of-speech tags).
          (S the dog chased the cat)
                                                                                          The order reflects the order of the leaves in the tree's
                                                                                hierarchical structure.
        return: a tree consisting of this tree's root connected directly
                                                                                        :rtype: list(tuple)
to
          its leaves, omitting all intervening non-terminal nodes.
                                                                                     pprint(self, margin=70, indent=0, nodesep=", parens='()',
       :rtype: Tree
                                                                                quotes=False)
                                                                                        :return: A pretty-printed string representation of this tree.
     freeze(self, leaf freezer=None)
                                                                                        :rtype: str
                                                                                        :param margin: The right margin at which to do line-wrapping.
     height(self)
                                                                                        :type margin: int
                                                                                        :param indent: The indentation level at which printing
       Return the height of the tree.
                                                                                           begins. This number is used to decide how far to indent
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                          subsequent lines.
                                                                                        :type indent: int
(D the) (N cat))))")
          >>> t.height()
                                                                                        :param nodesep: A string that is used to separate the node
                                                                                          from the children. E.g., the default value "": gives
                                                                                          trees like ``(S: (NP: I) (VP: (V: saw) (NP: it)))``.
          >>> print t[0,0]
          (D the)
          >>> t[0,0].height()
                                                                                     pprint_latex_qtree(self)
                                                                                        Returns a representation of the tree compatible with the
                                                                                        LaTeX gtree package. This consists of the string "\Tree"
       :return: The height of this tree. The height of a tree
                                                                                        followed by the parse tree represented in bracketed notation.
          containing no children is 1; the height of a tree
          containing only leaves is 2; and the height of any other
                                                                                        For example, the following result was generated from a parse
          tree is one plus the maximum of its children's
                                                                                tree of
          heights.
                                                                                        the sentence "The announcement astounded us"::
       :rtype: int
                                                                                         \Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
     leaf_treeposition(self, index)
                                                                                            [.l' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
       :return: The tree position of the ``index``-th leaf in this
          tree. I.e., if "tp=self.leaf treeposition(i)", then
                                                                                        See http://www.ling.upenn.edu/advice/latex.html for the
           ``self[tp]==self.leaves()[i]``.
                                                                                LaTeX
                                                                                        style file for the gtree package.
       :raise IndexError: If this tree contains fewer than ``index+1``
          leaves, or if ``index<0``.
                                                                                        :return: A latex gtree representation of this tree.
                                                                                        :rtype: str
     leaves(self)
       Return the leaves of the tree.
                                                                                     productions(self)
                                                                                        Generate the productions that correspond to the non-terminal
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                nodes of the tree.
                                                                                        For each subtree of the form (P: C1 C2 ... Cn) this produces a
(D the) (N cat))))")
          >>> t.leaves()
                                                                                production of the
          ['the', 'dog', 'chased', 'the', 'cat']
                                                                                        form P -> C1 C2 ... Cn.
       :return: a list containing this tree's leaves.
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
          The order reflects the order of the
                                                                                (D the) (N cat))))")
          leaves in the tree's hierarchical structure.
                                                                                          >>> t.productions()
                                                                                          [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP,
       :rtype: list
                                                                                V -> 'chased',
                                                                                          NP -> D N, D -> 'the', N -> 'cat']
     pos(self)
       Return a sequence of pos-tagged words extracted from the
tree.
                                                                                        :rtype: list(Production)
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                     subtrees(self, filter=None)
                                                                                        Generate all the subtrees of this tree, optionally restricted
(D the) (N cat))))")
                                                                                        to trees matching the filter function.
          [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
```

```
>>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                                Convert a tree between different subtypes of Tree. "cls"
(D the) (N cat))))")
                                                                                                                     determines
               >>> for s in t.subtrees(lambda t: t.height() == 2):
                                                                                                                                which class will be used to encode the new tree.
                      print s
               (D the)
                                                                                                                                :type tree: Tree
                                                                                                                                :param tree: The tree that should be converted.
               (N dog)
                                                                                                                                 :return: The new Tree.
               (V chased)
               (D the)
               (N cat)
                                                                                                                            parse(cls, s, brackets='()', parse node=None, parse leaf=None,
                                                                                                                     node pattern=None, leaf pattern=None,
                                                                                                                     remove empty top bracketing=False) from builtin .type
           :type filter: function
           :param filter: the function to filter all local trees
                                                                                                                                Parse a bracketed tree string and return the resulting tree.
                                                                                                                                Trees are represented as nested brackettings, such as::
       treeposition_spanning_leaves(self, start, end)
           :return: The tree position of the lowest descendant of this
                                                                                                                                  (S (NP (NNP John)) (VP (V runs)))
               tree that dominates ``self.leaves()[start:end]``.
           :raise ValueError: if ``end <= start``
                                                                                                                                :type s: str
                                                                                                                                 :param s: The string to parse
       treepositions(self, order='preorder')
               >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                                 :type brackets: str (length=2)
                                                                                                                                :param brackets: The bracket characters used to mark the
(D the) (N cat)))")
               >>> t.treepositions() # doctest: +ELLIPSIS
                                                                                                                                    beginning and end of trees and subtrees.
               [(), (0,), (0, 0), (0, 0, 0), (0, 1), (0, 1, 0), (1,), (1, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (
0), ...]
                                                                                                                                 :type parse_node: function
               >>> for pos in t.treepositions('leaves'):
                                                                                                                                 :type parse_leaf: function
               ... t[pos] = t[pos][::-1].upper()
                                                                                                                                 :param parse_node, parse_leaf: If specified, these functions
                                                                                                                                    are applied to the substrings of "s" corresponding to
               >>> print t
                                                                                                                                    nodes and leaves (respectively) to obtain the values for
               (S (NP (D EHT) (N GOD)) (VP (V DESAHC) (NP (D EHT)
                                                                                                                                    those nodes and leaves. They should have the following
(N TAC))))
                                                                                                                                    signature:
            :param order: One of: ``preorder``, ``postorder``, ``bothorder``,
                `leaves``.
                                                                                                                                       parse_node(str) -> value
       un chomsky normal form(self, expandUnary=True,
                                                                                                                                    For example, these functions could be used to parse nodes
childChar='|', parentChar='^', unaryChar='+')
                                                                                                                                    and leaves whose values should be some type other than
           This method modifies the tree in three ways:
                                                                                                                                    string (such as ``FeatStruct``).
                                                                                                                                    Note that by default, node strings and leaf strings are
             1. Transforms a tree in Chomsky Normal Form back to its
                                                                                                                                    delimited by whitespace and brackets; to override this
                original structure (branching greater than two)
                                                                                                                                    default, use the "node_pattern" and "leaf_pattern"
             2. Removes any parent annotation (if it exists)
                                                                                                                                    arguments.
             3. (optional) expands unary subtrees (if previously
                collapsed with collapseUnary(...))
                                                                                                                                 :type node_pattern: str
                                                                                                                                 :type leaf pattern: str
           :param expandUnary: Flag to expand unary or not (default =
                                                                                                                                 :param node_pattern, leaf_pattern: Regular expression
True)
                                                                                                                     patterns
           :type expandUnary: bool
                                                                                                                                    used to find node and leaf substrings in "s". By
           :param childChar: A string separating the head node from its
                                                                                                                                    default, both nodes patterns are defined to match any
children in an artificial node (default = "|")
                                                                                                                                    sequence of non-whitespace non-bracket characters.
           :type childChar: str
           :param parentChar: A sting separating the node label from its
                                                                                                                                 :type remove_empty_top_bracketing: bool
parent annotation (default = "^")
                                                                                                                                 :param remove empty top bracketing: If the resulting tree
           :type parentChar: str
                                                                                                                     has
           :param unaryChar: A string joining two non-terminals in a
                                                                                                                                    an empty node label, and is length one, then return its
unary production (default = "+")
                                                                                                                                    single child instead. This is useful for treebank trees,
           :type unaryChar: str
                                                                                                                                    which sometimes contain an extra level of bracketing.
                                                                                                                                :return: A tree corresponding to the string representation "s".
       Class methods inherited from Tree:
                                                                                                                                    If this class method is called using a subclass of Tree,
                                                                                                                                    then it will return a tree of that type.
       convert(cls, tree) from __builtin__.type
                                                                                                                                :rtype: Tree
```

```
T.__new__(S, ...) -> a new object with type S, a subtype of T
                                                                                  class ProbabilisticMixIn( builtin .object)
     Data descriptors inherited from Tree:
                                                                                    A mix-in class to associate probabilities with other classes
       dict
                                                                                    (trees, rules, etc.). To use the "ProbabilisticMixIn" class,
                                                                                    define a new class that derives from an existing class and from
       dictionary for instance variables (if defined)
                                                                                    ProbabilisticMixIn. You will need to define a new constructor for
                                                                                    the new class, which explicitly calls the constructors of both its
       weakref
       list of weak references to the object (if defined)
                                                                                     parent classes. For example:
                                                                                       >>> from nltk.probability import ProbabilisticMixIn
     Methods inherited from __builtin__.list:
                                                                                       >>> class A:
                                                                                            def \underline{\quad} init\underline{\quad} (self, x, y): self.data = (x,y)
       _contains__(...)
       x.__contains__(y) <==> y in x
                                                                                       >>> class ProbabilisticA(A, ProbabilisticMixIn):
                                                                                            def __init__(self, x, y, **prob_kwarg):
                                                                                               A.__init__(self, x, y)
     __getattribute__(...)
       x.__getattribute__('name') <==> x.name
                                                                                               ProbabilisticMixIn.__init__(self, **prob_kwarg)
                                                                                    See the documentation for the ProbabilisticMixIn
       _iadd___(...)
       x._iadd_(y) <==> x+=y
                                                                                     ``constructor< init >`` for information about the arguments it
                                                                                    expects.
      imul (...)
       x._imul_(y) <==> x*=y
                                                                                    You should generally also redefine the string representation
                                                                                    methods, the comparison methods, and the hashing method.
      iter__(...)
       x.__iter__() <==> iter(x)
                                                                                    Methods defined here:
                                                                                       _init__(self, **kwargs)
      __len__(...)
       x.__len__() <==> len(x)
                                                                                       Initialize this object's probability. This initializer should
                                                                                       be called by subclass constructors. "prob" should generally
     __reversed__(...)
                                                                               be
       L. reversed _() -- return a reverse iterator over the list
                                                                                       the first argument for those constructors.
     __sizeof__(...)
                                                                                       :param prob: The probability associated with the object.
       L.__sizeof__() -- size of L in memory, in bytes
                                                                                       :type prob: float
                                                                                       :param logprob: The log of the probability associated with
     count(...)
                                                                                          the object.
       L.count(value) -> integer -- return number of occurrences of
                                                                                       :type logprob: float
value
                                                                                    logprob(self)
                                                                                       Return ``log(p)``, where ``p`` is the probability associated
    index(...)
       L.index(value, [start, [stop]]) -> integer -- return first index of
                                                                                       with this object.
value.
       Raises ValueError if the value is not present.
                                                                                       :rtype: float
    reverse(...)
                                                                                    prob(self)
       L.reverse() -- reverse *IN PLACE*
                                                                                       Return the probability associated with this object.
    sort(...)
                                                                                       :rtype: float
       L.sort(cmp=None, key=None, reverse=False) -- stable sort
*IN PLACE*;
                                                                                    set logprob(self, logprob)
       cmp(x, y) -> -1, 0, 1
                                                                                       Set the log probability associated with this object to
                                                                                       "logprob". I.e., set the probability associated with this
                                                                                       object to ``2**(logprob)``.
     Data and other attributes inherited from builtin .list:
                                                                                       :param logprob: The new log probability
                                                                                       :type logprob: float
     hash = None
     __new__ = <built-in method __new__ of type object>
                                                                                    set prob(self, prob)
```

Set the probability associated with this object to ``prob``.	mul(self, v)
	j i
:param prob: The new probability :type prob: float	radd(self, v)
	rmul(self, v)
Data descriptors defined here:	setitem(self, index, value)
 dict	 chomsky_normal_form(self, factor='right', horzMarkov=None,
dictionary for instance variables (if defined)	vertMarkov=0, childChar=' ', parentChar='^')
weakref	This method can modify a tree in three ways:
list of weak references to the object (if defined)	1. Convert a tree into its Chomsky Normal Form (CNF)
along Dock of Walfa Tana / Tana and Jana and a fall to Dock of Walfa Maria A	equivalent Every subtree has either two non-terminals
class ProbabilisticTree(Tree, nltk.probability.ProbabilisticMixIn) Method resolution order:	or one terminal as its children. This process requires the creation of more artificial non-terminal nodes.
ProbabilisticTree	2. Markov (vertical) smoothing of children in new artificial
Tree	nodes
builtinlist	3. Horizontal (parent) annotation of nodes
nltk.probability.ProbabilisticMixIn	
builtinobject	<pre>param factor: Right or left factoring method (default = "right") :type factor: str = [left right]</pre>
Methods defined here:	:param horzMarkov: Markov order for sibling smoothing in
	artificial nodes (None (default) = include all siblings)
cmp(self, other)	:type horzMarkov: int None
 eq(self, other)	:param vertMarkov: Markov order for parent smoothing (0 (default) = no vertical annotation)
(3011, 301101)	:type vertMarkov: int None
init(self, node_or_str, children=None, **prob_kwargs)	:param childChar: A string used in construction of the artificial
	nodes, separating the head of the
ne(self, other)	original subtree from the child nodes that have yet
 repr(self)	to be expanded (default = " ") :type childChar: str
	:param parentChar: A string used to separate the node
str(self)	representation from its vertical annotation
copy(self, deep=False)	type parentChar: str
copy(sell, deep=raise)	l collapse_unary(self, collapsePOS=False, collapseRoot=False,
	joinChar='+')
Class methods defined here:	Collapse subtrees with a single child (ie. unary productions)
l convert(cls, val) frombuiltintype	into a new non-terminal (Tree node) joined by 'joinChar'. This is useful when working with algorithms that do not allow
	unary productions, and completely removing the unary
	productions
Methods inherited from Tree:	would require loss of useful information. The Tree is modified
 add(self, v)	directly (since it is passed by reference) and no value is returned.
delitem(self, index)	param collapsePOS: 'False' (default) will not collapse the
 <u>ge(self, other)</u>	parent of leaf nodes (ie. Part-of-Speech tags) since they are always unary
ge(sell, other) 	productions
getitem(self, index)	:type collapsePOS: bool
l at (alf allow)	param collapseRoot: 'False' (default) will not modify the root
gt(self, other)	production if it is unary. For the Penn WSJ treebank corpus
lle(self, other)	this corresponds
ĺ	to the TOP -> productions.
lt(self, other)	:type collapseRoot: bool

```
:param joinChar: A string used to connect collapsed node
                                                                                           The order reflects the order of the
values (default = "+")
                                                                                          leaves in the tree's hierarchical structure.
        :type joinChar: str
                                                                                        :rtype: list
    draw(self)
                                                                                     pos(self)
       Open a new window containing a graphical diagram of this
                                                                                        Return a sequence of pos-tagged words extracted from the
tree.
                                                                                tree.
     flatten(self)
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
       Return a flat version of the tree, with all non-root non-
                                                                                (D the) (N cat))))")
terminals removed.
                                                                                          >>> t.pos()
                                                                                          [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
(D the) (N cat))))")
                                                                                        :return: a list of tuples containing leaves and pre-terminals
          >>> print t.flatten()
                                                                                (part-of-speech tags).
          (S the dog chased the cat)
                                                                                          The order reflects the order of the leaves in the tree's
                                                                                hierarchical structure.
                                                                                        :rtype: list(tuple)
        :return: a tree consisting of this tree's root connected directly
to
          its leaves, omitting all intervening non-terminal nodes.
                                                                                    | pprint(self, margin=70, indent=0, nodesep=", parens='()',
       :rtype: Tree
                                                                                quotes=False)
                                                                                        :return: A pretty-printed string representation of this tree.
    freeze(self, leaf_freezer=None)
                                                                                        :rtvpe: str
                                                                                        :param margin: The right margin at which to do line-wrapping.
    height(self)
                                                                                        :type margin: int
       Return the height of the tree.
                                                                                        :param indent: The indentation level at which printing
                                                                                          begins. This number is used to decide how far to indent
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                           subsequent lines.
                                                                                        :type indent: int
(D the) (N cat))))")
          >>> t.height()
                                                                                        :param nodesep: A string that is used to separate the node
                                                                                          from the children. E.g., the default value "':" gives
                                                                                          trees like ``(S: (NP: I) (VP: (V: saw) (NP: it)))``.
          >>> print t[0,0]
          (D the)
          >>> t[0,0].height()
                                                                                     pprint latex gtree(self)
                                                                                        Returns a representation of the tree compatible with the
                                                                                        LaTeX gtree package. This consists of the string "\Tree"
       :return: The height of this tree. The height of a tree
                                                                                        followed by the parse tree represented in bracketed notation.
          containing no children is 1; the height of a tree
          containing only leaves is 2; and the height of any other
                                                                                        For example, the following result was generated from a parse
          tree is one plus the maximum of its children's
                                                                                tree of
          heights.
                                                                                        the sentence "The announcement astounded us"::
       :rtype: int
                                                                                         \Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
     leaf treeposition(self, index)
                                                                                            [.I' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
       :return: The tree position of the ``index``-th leaf in this
          tree. I.e., if ``tp=self.leaf_treeposition(i)``, then
                                                                                        See http://www.ling.upenn.edu/advice/latex.html for the
          ``self[tp]==self.leaves()[i]``.
                                                                                LaTeX
                                                                                        style file for the qtree package.
       :raise IndexError: If this tree contains fewer than ``index+1``
          leaves, or if ``index<0``.
                                                                                        :return: A latex gtree representation of this tree.
                                                                                        :rtype: str
     leaves(self)
       Return the leaves of the tree.
                                                                                     productions(self)
                                                                                        Generate the productions that correspond to the non-terminal
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                nodes of the tree.
(D the) (N cat))))")
                                                                                        For each subtree of the form (P: C1 C2 ... Cn) this produces a
          >>> t.leaves()
                                                                                production of the
          ['the', 'dog', 'chased', 'the', 'cat']
                                                                                        form P -> C1 C2 ... Cn.
       :return: a list containing this tree's leaves.
```

```
>>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                      :param childChar: A string separating the head node from its
(D the) (N cat))))")
                                                                              children in an artificial node (default = "|")
                                                                                      :type childChar: str
          >>> t.productions()
                                                                                      :param parentChar: A sting separating the node label from its
          [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP,
V -> 'chased',
                                                                              parent annotation (default = "^")
                                                                                      :type parentChar: str
          NP -> D N, D -> 'the', N -> 'cat']
                                                                                      :param unaryChar: A string joining two non-terminals in a
       :rtype: list(Production)
                                                                              unary production (default = "+")
                                                                                      :type unaryChar: str
    subtrees(self, filter=None)
       Generate all the subtrees of this tree, optionally restricted
       to trees matching the filter function.
                                                                                   Class methods inherited from Tree:
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                   parse(cls, s, brackets='()', parse_node=None, parse_leaf=None,
(D the) (N cat))))")
                                                                              node pattern=None, leaf pattern=None,
          >>> for s in t.subtrees(lambda t: t.height() == 2):
                                                                              remove empty top bracketing=False) from builtin .type
                                                                                      Parse a bracketed tree string and return the resulting tree.
          ... print s
                                                                                      Trees are represented as nested brackettings, such as::
          (D the)
          (N dog)
          (V chased)
                                                                                       (S (NP (NNP John)) (VP (V runs)))
          (D the)
          (N cat)
                                                                                      :type s: str
                                                                                      :param s: The string to parse
       :type filter: function
       :param filter: the function to filter all local trees
                                                                                      :type brackets: str (length=2)
                                                                                      :param brackets: The bracket characters used to mark the
    treeposition_spanning_leaves(self, start, end)
                                                                                        beginning and end of trees and subtrees.
       :return: The tree position of the lowest descendant of this
          tree that dominates ``self.leaves()[start:end]``.
                                                                                      :type parse_node: function
       :raise ValueError: if ``end <= start``
                                                                                      :type parse_leaf: function
                                                                                      :param parse_node, parse_leaf: If specified, these functions
                                                                                        are applied to the substrings of "s" corresponding to
    treepositions(self, order='preorder')
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                        nodes and leaves (respectively) to obtain the values for
                                                                                        those nodes and leaves. They should have the following
(D the) (N cat))))")
          >>> t.treepositions() # doctest: +ELLIPSIS
                                                                                        signature:
          [(), (0,), (0, 0), (0, 0, 0), (0, 1), (0, 1, 0), (1,), (1, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0)]
0), ...]
                                                                                          parse_node(str) -> value
          >>> for pos in t.treepositions('leaves'):
          ... t[pos] = t[pos][::-1].upper()
                                                                                        For example, these functions could be used to parse nodes
         >>> print t
                                                                                        and leaves whose values should be some type other than
          (S (NP (D EHT) (N GOD)) (VP (V DESAHC) (NP (D EHT)
                                                                                        string (such as ``FeatStruct``).
                                                                                        Note that by default, node strings and leaf strings are
(N TAC))))
                                                                                        delimited by whitespace and brackets; to override this
                                                                                        default, use the ``node_pattern`` and ``leaf_pattern``
        :param order: One of: ``preorder``, ``postorder``, ``bothorder``,
          ``leaves``.
                                                                                        arguments.
    un chomsky normal form(self, expandUnary=True,
                                                                                      :type node pattern: str
childChar='|', parentChar='^', unaryChar='+')
                                                                                      :type leaf_pattern: str
       This method modifies the tree in three ways:
                                                                                      :param node_pattern, leaf_pattern: Regular expression
                                                                              patterns
         1. Transforms a tree in Chomsky Normal Form back to its
                                                                                        used to find node and leaf substrings in "s". By
          original structure (branching greater than two)
                                                                                        default, both nodes patterns are defined to match any
         2. Removes any parent annotation (if it exists)
                                                                                        sequence of non-whitespace non-bracket characters.
         3. (optional) expands unary subtrees (if previously
          collapsed with collapseUnary(...))
                                                                                      :type remove_empty_top_bracketing: bool
                                                                                      :param remove_empty_top_bracketing: If the resulting tree
        :param expandUnary: Flag to expand unary or not (default =
                                                                              has
True)
                                                                                        an empty node label, and is length one, then return its
       :type expandUnary: bool
                                                                                        single child instead. This is useful for treebank trees,
                                                                                        which sometimes contain an extra level of bracketing.
```

```
count(...)
  :return: A tree corresponding to the string representation "s".
                                                                                  L.count(value) -> integer -- return number of occurrences of
     If this class method is called using a subclass of Tree,
                                                                          value
     then it will return a tree of that type.
  :rtype: Tree
                                                                               extend(...)
                                                                                 L.extend(iterable) -- extend list by appending elements from
                                                                          the iterable
Data descriptors inherited from Tree:
                                                                               index(...)
  dict
                                                                                  L.index(value, [start, [stop]]) -> integer -- return first index of
  dictionary for instance variables (if defined)
                                                                          value.
                                                                                  Raises ValueError if the value is not present.
  weakref
  list of weak references to the object (if defined)
                                                                               insert(...)
                                                                                  L.insert(index, object) -- insert object before index
Methods inherited from builtin .list:
                                                                               pop(...)
                                                                                  L.pop([index]) -> item -- remove and return item at index
  contains__(...)
                                                                          (default last).
  x._contains__(y) <==> y in x
                                                                                  Raises IndexError if list is empty or index is out of range.
 _delslice__(...)
                                                                               remove(...)
  x.__delslice__(i, j) <==> del x[i:j]
                                                                                  L.remove(value) -- remove first occurrence of value.
                                                                                  Raises ValueError if the value is not present.
  Use of negative indices is not supported.
                                                                               reverse(...)
 getattribute (...)
                                                                                  L.reverse() -- reverse *IN PLACE*
  x. getattribute ('name') <==> x.name
                                                                               sort(...)
                                                                                  L.sort(cmp=None, key=None, reverse=False) -- stable sort
  _getslice__(...)
  x.__getslice__(i, j) <==> x[i:j]
                                                                          *IN PLACE*:
                                                                                  cmp(x, y) \rightarrow -1, 0, 1
  Use of negative indices is not supported.
                                                                               Data and other attributes inherited from __builtin__.list:
  _iadd___(...)
  x._iadd_(y) <==> x+=y
                                                                               hash = None
 _imul__(...)
  x._imul_(y) <==> x*=y
                                                                               __new__ = <built-in method __new__ of type object>
                                                                                  T. new (S, ...) -> a new object with type S, a subtype of T
 _iter__(...)
  x.__iter__() <==> iter(x)
                                                                               Methods inherited from nltk.probability.ProbabilisticMixIn:
__len__(...)
  x._len_() <==> len(x)
                                                                               logprob(self)
                                                                                  Return ``log(p)``, where ``p`` is the probability associated
 reversed (...)
                                                                                  with this object.
  L.__reversed__() -- return a reverse iterator over the list
                                                                                  :rtype: float
 _setslice__(...)
  x.__setslice__(i, j, y) <==> x[i:j]=y
                                                                                  Return the probability associated with this object.
  Use of negative indices is not supported.
                                                                                  :rtype: float
  _sizeof___(...)
  L.__sizeof__() -- size of L in memory, in bytes
                                                                               set_logprob(self, logprob)
                                                                                  Set the log probability associated with this object to
                                                                                  "logprob". I.e., set the probability associated with this
append(...)
                                                                                  object to ``2**(logprob)``.
  L.append(object) -- append object to end
```

```
:param logprob: The new log probability
       :type logprob: float
                                                                                   I.e., every tree position is either a single index *i*,
                                                                                   specifying "tree[i]"; or a sequence *i1, i2, ..., iN*,
    set_prob(self, prob)
                                                                                   specifying "tree[i1][i2]...[iN].
       Set the probability associated with this object to "prob".
                                                                                   Construct a new tree. This constructor can be called in one
                                                                                   of two ways:
       :param prob: The new probability
       :type prob: float
                                                                                   - "Tree(node, children)" constructs a new tree with the
                                                                                      specified node value and list of children.
  class Tree( builtin .list)
  A Tree represents a hierarchical grouping of leaves and
subtrees.
                                                                                   - "Tree(s)" constructs a new tree by parsing the string "s".
   | For example, each constituent in a syntax tree is represented by
                                                                                      It is equivalent to calling the class method "Tree.parse(s)".
a single Tree.
                                                                                   Method resolution order:
    A tree's children are encoded as a list of leaves and subtrees,
                                                                                      Tree
     where a leaf is a basic (non-tree) value; and a subtree is a
                                                                                      builtin .list
     nested Tree.
                                                                                      __builtin__.object
                                                                                   Methods defined here:
       >>> from nltk.tree import Tree
       >>> print Tree(1, [2, Tree(3, [4]), 5])
       (12(34)5)
                                                                                   __add__(self, v)
       >>> vp = Tree('VP', [Tree('V', ['saw']),
                     Tree('NP', ['him'])])
                                                                                     _delitem__(self, index)
       >>> s = Tree('S', [Tree('NP', ['I']), vp])
       >>> print s
                                                                                    __eq__(self, other)
       (S (NP I) (VP (V saw) (NP him)))
       >>> print s[1]
                                                                                    __ge__(self, other)
       (VP (V saw) (NP him))
       >>> print s[1,1]
                                                                                     _getitem__(self, index)
       (NP him)
       >>> t = Tree("(S (NP I) (VP (V saw) (NP him)))")
                                                                                    __gt__(self, other)
       >>> s == t
       True
                                                                                    __init__(self, node_or_str, children=None)
       >>> t[1][1].node = "X"
       >>> print t
                                                                                   __le__(self, other)
       (S (NP I) (VP (V saw) (X him)))
       >>> t[0], t[1,1] = t[1,1], t[0]
                                                                                    __lt__(self, other)
       >>> print t
       (S (X him) (VP (V saw) (NP I)))
                                                                                   mul (self, v)
     The length of a tree is the number of children it has.
                                                                                   __ne__(self, other)
       >>> len(t)
                                                                                    __radd__(self, v)
                                                                                   __repr__(self)
     Any other properties that a Tree defines are known as node
     properties, and are used to add information about individual
                                                                                    __rmul__(self, v)
     hierarchical groupings. For example, syntax trees use a NODE
     property to label syntactic constituents with phrase tags, such as
                                                                                    _setitem__(self, index, value)
     "NP" and "VP".
                                                                                   str (self)
     Several Tree methods use "tree positions" to specify
     children or descendants of a tree. Tree positions are defined as
                                                                                   chomsky_normal_form(self, factor='right', horzMarkov=None,
     follows:
                                                                              vertMarkov=0, childChar='|', parentChar='^')
                                                                                      This method can modify a tree in three ways:
      - The tree position *i* specifies a Tree's *i*\ th child.
      - The tree position ``()`` specifies the Tree itself.
                                                                                       1. Convert a tree into its Chomsky Normal Form (CNF)
      - If *p* is the tree position of descendant *d*, then
```

p+i specifies the *i*\ th child of *d*.

equivalent -- Every subtree has either two non-terminals or one terminal as its children. This process requires

```
the creation of more artificial non-terminal nodes.
                                                                                          >>> print t.flatten()
                                                                                          (S the dog chased the cat)
         2. Markov (vertical) smoothing of children in new artificial
           nodes
         3. Horizontal (parent) annotation of nodes
                                                                                       :return: a tree consisting of this tree's root connected directly
                                                                               to
        :param factor: Right or left factoring method (default = "right")
                                                                                          its leaves, omitting all intervening non-terminal nodes.
        :type factor: str = [left|right]
                                                                                        :rtvpe: Tree
        :param horzMarkov: Markov order for sibling smoothing in
artificial nodes (None (default) = include all siblings)
                                                                                     freeze(self, leaf freezer=None)
        :type horzMarkov: int | None
        :param vertMarkov: Markov order for parent smoothing (0
                                                                                     height(self)
(default) = no vertical annotation)
                                                                                       Return the height of the tree.
        :type vertMarkov: int | None
        :param childChar: A string used in construction of the artificial
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
nodes, separating the head of the
                                                                               (D the) (N cat))))")
                   original subtree from the child nodes that have yet
                                                                                          >>> t.height()
to be expanded (default = "|")
        :type childChar: str
                                                                                          >>> print t[0,0]
        :param parentChar: A string used to separate the node
                                                                                          (D the)
representation from its vertical annotation
                                                                                          >>> t[0,0].height()
        :type parentChar: str
   collapse unary(self, collapsePOS=False, collapseRoot=False,
                                                                                        :return: The height of this tree. The height of a tree
ioinChar='+')
                                                                                          containing no children is 1; the height of a tree
       Collapse subtrees with a single child (ie. unary productions)
                                                                                          containing only leaves is 2; and the height of any other
       into a new non-terminal (Tree node) joined by 'joinChar'.
                                                                                          tree is one plus the maximum of its children's
        This is useful when working with algorithms that do not allow
                                                                                          heights.
                                                                                       :rtype: int
        unary productions, and completely removing the unary
productions
        would require loss of useful information. The Tree is modified
                                                                                    leaf_treeposition(self, index)
       directly (since it is passed by reference) and no value is
                                                                                       :return: The tree position of the ``index``-th leaf in this
                                                                                          tree. I.e., if ``tp=self.leaf_treeposition(i)``, then
returned.
                                                                                          ``self[tp]==self.leaves()[i]``.
        :param collapsePOS: 'False' (default) will not collapse the
parent of leaf nodes (ie.
                                                                                       :raise IndexError: If this tree contains fewer than ``index+1``
                     Part-of-Speech tags) since they are always unary
                                                                                          leaves, or if ``index<0``.
productions
        :type collapsePOS: bool
                                                                                     leaves(self)
        :param collapseRoot: 'False' (default) will not modify the root
                                                                                       Return the leaves of the tree.
production
                     if it is unary. For the Penn WSJ treebank corpus,
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                (D the) (N cat))))")
this corresponds
                     to the TOP -> productions.
                                                                                          >>> t.leaves()
        :type collapseRoot: bool
                                                                                          ['the', 'dog', 'chased', 'the', 'cat']
        :param joinChar: A string used to connect collapsed node
values (default = "+")
                                                                                       :return: a list containing this tree's leaves.
                                                                                          The order reflects the order of the
        :type joinChar: str
                                                                                          leaves in the tree's hierarchical structure.
     copy(self, deep=False)
                                                                                       :rtype: list
        Open a new window containing a graphical diagram of this
                                                                                       Return a sequence of pos-tagged words extracted from the
                                                                               tree.
tree.
     flatten(self)
                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
        Return a flat version of the tree, with all non-root non-
                                                                               (D the) (N cat))))")
terminals removed.
                                                                                          >>> t.pos()
                                                                                          [('the', 'D'), ('dog', 'N'), ('chased', 'V'), ('the', 'D'), ('cat', 'N')]
          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
(D the) (N cat))))")
```

```
:return: a list of tuples containing leaves and pre-terminals
                                                                                                                                         >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
(part-of-speech tags).
                                                                                                                         (D the) (N cat))))")
               The order reflects the order of the leaves in the tree's
                                                                                                                                         >>> for s in t.subtrees(lambda t: t.height() == 2):
hierarchical structure.
                                                                                                                                                 print s
           :rtype: list(tuple)
                                                                                                                                          (D the)
                                                                                                                                          (N dog)
     | pprint(self, margin=70, indent=0, nodesep=", parens='()',
                                                                                                                                          (V chased)
quotes=False)
                                                                                                                                          (D the)
           :return: A pretty-printed string representation of this tree.
                                                                                                                                          (N cat)
           :param margin: The right margin at which to do line-wrapping.
                                                                                                                                      :type filter: function
                                                                                                                                      :param filter: the function to filter all local trees
           :type margin: int
           :param indent: The indentation level at which printing
               begins. This number is used to decide how far to indent
                                                                                                                                 treeposition_spanning_leaves(self, start, end)
               subsequent lines.
                                                                                                                                      :return: The tree position of the lowest descendant of this
                                                                                                                                          tree that dominates ``self.leaves()[start:end]``.
            :type indent: int
            :param nodesep: A string that is used to separate the node
                                                                                                                                      :raise ValueError: if ``end <= start``
               from the children. E.g., the default value ``':'`` gives
               trees like "(S: (NP: I) (VP: (V: saw) (NP: it)))".
                                                                                                                                 treepositions(self, order='preorder')
                                                                                                                                          >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
       pprint latex gtree(self)
                                                                                                                         (D the) (N cat)))")
           Returns a representation of the tree compatible with the
                                                                                                                                         >>> t.treepositions() # doctest: +ELLIPSIS
                                                                                                                                         [(), (0,), (0, 0), (0, 0, 0), (0, 1), (0, 1, 0), (1,), (1, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (1, 0, 0), (
           LaTeX gtree package. This consists of the string "\Tree"
           followed by the parse tree represented in bracketed notation.
                                                                                                                          0), ...]
                                                                                                                                         >>> for pos in t.treepositions('leaves'):
                                                                                                                                         ... t[pos] = t[pos][::-1].upper()
           For example, the following result was generated from a parse
tree of
                                                                                                                                         >>> print t
           the sentence "The announcement astounded us"::
                                                                                                                                         (S (NP (D EHT) (N GOD)) (VP (V DESAHC) (NP (D EHT)
                                                                                                                         (N TAC))))
             \Tree [.I" [.N" [.D The ] [.N' [.N announcement ] ] ]
                 [.I' [.V" [.V' [.V astounded ] [.N" [.N' [.N us ] ] ] ] ] ]
                                                                                                                                      :param order: One of: ``preorder``, ``postorder``, ``bothorder``,
                                                                                                                                          `leaves``.
           See http://www.ling.upenn.edu/advice/latex.html for the
                                                                                                                                 un chomsky normal form(self, expandUnary=True,
LaTeX
           style file for the qtree package.
                                                                                                                         childChar='|', parentChar='^', unaryChar='+')
                                                                                                                                     This method modifies the tree in three ways:
           :return: A latex gtree representation of this tree.
                                                                                                                                        1. Transforms a tree in Chomsky Normal Form back to its
           :rtype: str
                                                                                                                                          original structure (branching greater than two)
                                                                                                                                        2. Removes any parent annotation (if it exists)
       productions(self)
                                                                                                                                        3. (optional) expands unary subtrees (if previously
           Generate the productions that correspond to the non-terminal
nodes of the tree.
                                                                                                                                          collapsed with collapseUnary(...))
           For each subtree of the form (P: C1 C2 ... Cn) this produces a
production of the
                                                                                                                                      :param expandUnary: Flag to expand unary or not (default =
           form P -> C1 C2 ... Cn.
                                                                                                                         True)
                                                                                                                                     :type expandUnary: bool
                                                                                                                                      :param childChar: A string separating the head node from its
               >>> t = Tree("(S (NP (D the) (N dog)) (VP (V chased) (NP
                                                                                                                         children in an artificial node (default = "|")
(D the) (N cat))))")
               >>> t.productions()
                                                                                                                                     :type childChar: str
               [S -> NP VP, NP -> D N, D -> 'the', N -> 'dog', VP -> V NP,
                                                                                                                                      :param parentChar: A sting separating the node label from its
                                                                                                                         parent annotation (default = "^")
V -> 'chased'.
                                                                                                                                     :type parentChar: str
               NP -> D N, D -> 'the', N -> 'cat']
                                                                                                                                      :param unaryChar: A string joining two non-terminals in a
           :rtype: list(Production)
                                                                                                                         unary production (default = "+")
                                                                                                                                      :type unaryChar: str
       subtrees(self, filter=None)
           Generate all the subtrees of this tree, optionally restricted
           to trees matching the filter function.
                                                                                                                                 Class methods defined here:
```

convert(cls, tree) from __builtin__.type

```
Convert a tree between different subtypes of Tree. "cls"
determines
       which class will be used to encode the new tree.
                                                                                  Data descriptors defined here:
       :type tree: Tree
                                                                                    dict
       :param tree: The tree that should be converted.
                                                                                     dictionary for instance variables (if defined)
       :return: The new Tree.
                                                                                     weakref
    parse(cls, s, brackets='()', parse node=None, parse leaf=None,
                                                                                     list of weak references to the object (if defined)
node pattern=None, leaf pattern=None,
remove empty top bracketing=False) from builtin .type
       Parse a bracketed tree string and return the resulting tree.
                                                                                  Methods inherited from __builtin__.list:
       Trees are represented as nested brackettings, such as::
                                                                                    _contains__(...)
        (S (NP (NNP John)) (VP (V runs)))
                                                                                     x.__contains__(y) <==> y in x
                                                                                    _delslice__(...)
       :type s: str
                                                                                     x._delslice_(i, j) \le del x[i:j]
       :param s: The string to parse
       :type brackets: str (length=2)
                                                                                     Use of negative indices is not supported.
       :param brackets: The bracket characters used to mark the
          beginning and end of trees and subtrees.
                                                                                    _getattribute__(...)
                                                                                     x. getattribute ('name') <==> x.name
       :type parse_node: function
       :type parse_leaf: function
                                                                                    _getslice__(...)
       :param parse_node, parse_leaf: If specified, these functions
                                                                                     x.__getslice__(i, j) <==> x[i:j]
          are applied to the substrings of "s" corresponding to
          nodes and leaves (respectively) to obtain the values for
                                                                                     Use of negative indices is not supported.
          those nodes and leaves. They should have the following
          signature:
                                                                                    _iadd___(...)
                                                                                    x._iadd_(y) <==> x+=y
           parse_node(str) -> value
                                                                                    imul (...)
                                                                                    x. imul (y) <==> x*=y
          For example, these functions could be used to parse nodes
          and leaves whose values should be some type other than
          string (such as ``FeatStruct``).
                                                                                   __iter__(...)
          Note that by default, node strings and leaf strings are
                                                                                    x.__iter__() <==> iter(x)
          delimited by whitespace and brackets; to override this
          default, use the "node_pattern" and "leaf_pattern"
                                                                                    _len__(...)
                                                                                     x.__len__() <==> len(x)
          arguments.
       :type node_pattern: str
                                                                                   __reversed__(...)
       :type leaf pattern: str
                                                                                     L.__reversed__() -- return a reverse iterator over the list
       :param node_pattern, leaf_pattern: Regular expression
patterns
                                                                                    _setslice__(...)
                                                                                    x.__setslice__(i, j, y) <==> x[i:j]=y
          used to find node and leaf substrings in "s". By
          default, both nodes patterns are defined to match any
          sequence of non-whitespace non-bracket characters.
                                                                                     Use of negative indices is not supported.
       :type remove_empty_top_bracketing: bool
                                                                                    _sizeof__(...)
       :param remove empty top bracketing: If the resulting tree
                                                                                     L. sizeof () -- size of L in memory, in bytes
has
          an empty node label, and is length one, then return its
                                                                                  append(...)
          single child instead. This is useful for treebank trees,
                                                                                    L.append(object) -- append object to end
          which sometimes contain an extra level of bracketing.
       :return: A tree corresponding to the string representation "s".
                                                                                     L.count(value) -> integer -- return number of occurrences of
          If this class method is called using a subclass of Tree,
                                                                             value
          then it will return a tree of that type.
       :rtype: Tree
                                                                                  extend(...)
```

```
L.extend(iterable) -- extend list by appending elements from
the iterable
    index(...)
       L.index(value, [start, [stop]]) -> integer -- return first index of
value.
       Raises ValueError if the value is not present.
    insert(...)
       L.insert(index, object) -- insert object before index
    pop(...)
       L.pop([index]) -> item -- remove and return item at index
(default last).
       Raises IndexError if list is empty or index is out of range.
    remove(...)
       L.remove(value) -- remove first occurrence of value.
       Raises ValueError if the value is not present.
    reverse(...)
       L.reverse() -- reverse *IN PLACE*
    sort(...)
       L.sort(cmp=None, key=None, reverse=False) -- stable sort
*IN PLACE*;
       cmp(x, y) \rightarrow -1, 0, 1
    Data and other attributes inherited from __builtin__.list:
     __hash__ = None
      _new__ = <built-in method __new__ of type object>
       T.__new__(S, ...) -> a new object with type S, a subtype of T
FUNCTIONS
  bracket_parse(s)
    Use Tree.parse(s, remove_empty_top_bracketing=True) instead.
  sinica_parse(s)
    Parse a Sinica Treebank string and return a tree. Trees are
represented as nested brackettings.
     as shown in the following example (X represents a Chinese
character):
S(goal:NP(Head:Nep:XX)|theme:NP(Head:Nhaa:X)|quantity:Dab:X|H
ead:VL2:X)#0(PERIODCATEGORY)
     :return: A tree corresponding to the string representation.
     :rtype: Tree
     :param s: The string to be converted
    :type s: str
DATA
    _all__ = ['ImmutableProbabilisticTree', 'ImmutableTree',
'Probabilist...
```