

---

# **ctrie Documentation**

***Release 0.1.1***

**Baptiste Fontaine**

**Jun 09, 2016**



<b>1</b>	<b>Install</b>	<b>3</b>
<b>2</b>	<b>Guide</b>	<b>5</b>
2.1	API Reference . . . . .	5
2.2	Changes . . . . .	7
	<b>Python Module Index</b>	<b>9</b>



`ctrie` is a compact trie implementation for Python.



---

## Install

---

With `pip`:

```
[sudo] pip install ctrie
```





## 2.1 API Reference

### 2.1.1 `ctrie`

This module provides a compact trie (`CTrie`) implementation for Python.

**class** `ctrie.CTrie` (*terminal=False*)

A compact trie. A trie, or prefix tree, is a data structure to efficiently store a set of strings with a lot of shared prefixes, like URLs.

Compact tries are recursively defined, a trie has zero or more other child tries. A trie is said to be terminal if it contains the empty string. Each child is labelled with a prefix for its own children. For example, one compact trie that contains words “foo”, “bar” and “foo42” has two children, one labelled “foo” and the other labelled “bar”. The last one is terminal because once we read “bar” we can end our walk in the trie. The first one is terminal for the same reason, but have one terminal child labelled “42”. In the following symbolic representation, a terminated child is represented with a dot at the end of its label:

```
<empty>
|
+-- bar.
|
+-- foo.
    |
    +-- 42.
```

Each possible walk in the trie, starting from the root and ending on a terminal node is a contained word. In the previous example, the empty string is not present in the trie because the root node is not terminal.

A non-compact trie contains only one char per node.

```
>>> from ctrie import CTrie
>>> ct = CTrie()
>>> c.add("foo", "bar", "qux")
>>> "foo" in c
True
>>> "Bar" in c
False
>>> c.remove("foo")
>>> "foo" in c
False
```

```
>>> len(c)
2
```

**Keyword arguments:**

- `terminal` (bool, default: False): specifies if the trie contains the empty string. This is the same as:

```
>>> ct = CTrie()
>>> ct.add('') # empty string
```

**add(\*words)**

Add one or more words in the trie. The function returns `True` if all words were successfully added. It'll return `False` if one or more words were already present in the trie.

```
>>> ct = CTrie()
>>> ct.add('foo', 'bar')
True
>>> 'foo' in ct
True
>>> ct.add('bar')
False
>>> ct.add('qux')
True
```

**height()**

Compute the height of the trie. An empty trie has a zero height, and the maximum height of a compacted trie is the length of its longest word.

```
>>> ct = CTrie()
>>> ct.height()
0
>>> ct.add(''); ct.height()
0
>>> ct.add('foo'); ct.height()
1
>>> ct.add('bar'); ct.height()
1
>>> ct.add('boo'); ct.height()
2
```

New in version 0.1.0.

**is\_empty()**

Return `True` if the trie is empty.

```
>>> ct = CTrie()
>>> ct.is_empty()
True
>>> ct.add('')
True
>>> ct.is_empty()
False
```

**remove(\*words)**

Remove one or more words from the trie. The function returns `True` if all the words were successfully removed (i.e. they were in the trie before) or `False` if one or more words couldn't be found in the trie.

```
>>> ct = CTrie()
>>> ct.add('foo', 'qux')
True
>>> 'foo' in ct
True
>>> ct.remove('foo', 'bar')
False
>>> 'foo' in ct
False
>>> ct.remove('qux')
>>> True
```

### **values()**

Yield all values from this trie in arbitrary order. The order in which they're yielded doesn't depend on the order in which they're inserted.

```
>>> ct = CTrie()
>>> ct.add('foo', 'bar', 'qux')
True
>>> for x in ct.values(): print x
...
qux
foo
bar
```

New in version 0.1.0.

## 2.2 Changes

### 2.2.1 v0.1.1

- `__iter__` added
- `__iadd__` implemented, allowing you to extend a trie with `+=` and an iterable

### 2.2.2 v0.1.0

- memory footprint reduced
- `len` support
- `CTrie#height` and `CTrie#values` methods added

### 2.2.3 v0.0.1

- initial release
- support for add and remove operations
- full online documentation



## C

`ctrie`, 5



## A

`add()` (`ctrie.CTrie` method), 6

## C

`CTrie` (class in `ctrie`), 5

`ctrie` (module), 5

## H

`height()` (`ctrie.CTrie` method), 6

## I

`is_empty()` (`ctrie.CTrie` method), 6

## R

`remove()` (`ctrie.CTrie` method), 6

## V

`values()` (`ctrie.CTrie` method), 7