Any Python Tree Data

pypi package 2.8.0 | pypi downloads 632k/month | build passing | coverage 100% | docs passing
maintainability A | python 2.7 \| 3.4 \| 3.5 \| 3.6 \| 3.7 \| 3.8 | code style pep8 | code style pep257

Simple, lightweight and extensible Tree data structure.

Feel free to share info about your anytree project.

Links

- Documentation
- GitHub
- PyPI
- Changelog
- Issues
- Contributors
- If you enjoy anytree

  Buy me a coffee

Feel free to share info about your anytree project.

Getting started

Usage is simple.

v: latest

Construction

```
>>> from anytree import Node, RenderTree
>>> udo = Node("Udo")
>>> marc = Node("Marc", parent=udo)
>>> lian = Node("Lian", parent=marc)
>>> dan = Node("Dan", parent=udo)
>>> jet = Node("Jet", parent=dan)
>>> jan = Node("Jan", parent=dan)
>>> joe = Node("Joe", parent=dan)
```

Node

```
>>> print(udo)
Node('/Udo')
>>> print(joe)
Node('/Udo/Dan/Joe')
```

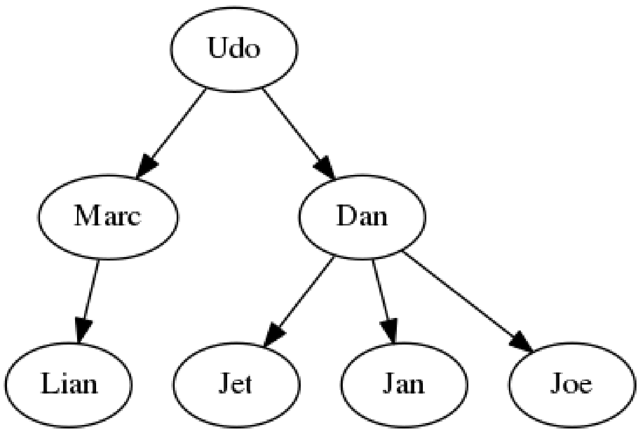
Tree

```
>>> for pre, fill, node in RenderTree(udo):
...     print("%s%s" % (pre, node.name))
Udo
├── Marc
│   └── Lian
└── Dan
    ├── Jet
    ├── Jan
    └── Joe
```

For details see `Node` and `RenderTree`.

Visualization

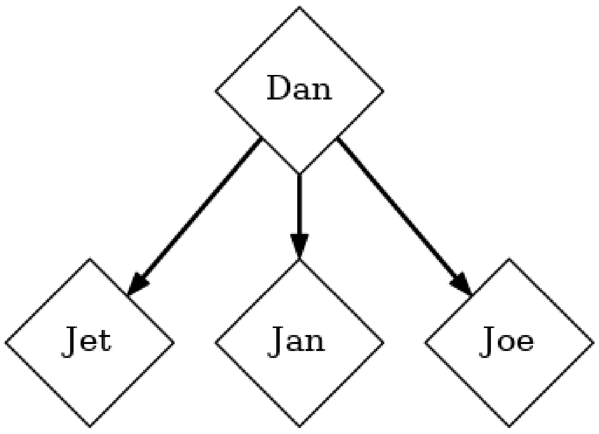
```
>>> from anytree.exporter import DotExporter
>>> # graphviz needs to be installed for the next line!
>>> DotExporter(udo).to_picture("udo.png")
```



The `DotExporter` can be started at any node and has various formatting hookups:

```
>>> DotExporter(dan,
...             nodeattrfunc=lambda node: "fixedsize=true, width=1, height=1, shape=diamond"
...             edgeattrfunc=lambda parent, child: "style=bold"
... ).to_picture("dan.png")
```

v: latest



Manipulation

A second tree:

```
>>> mary = Node("Mary")
>>> urs = Node("Urs", parent=mary)
>>> chris = Node("Chris", parent=mary)
>>> marta = Node("Marta", parent=mary)
>>> print(RenderTree(mary))
Node('/Mary')
├── Node('/Mary/Urs')
├── Node('/Mary/Chris')
└── Node('/Mary/Marta')
```

Append:

```
>>> udo.parent = mary
>>> print(RenderTree(mary))
Node('/Mary')
├── Node('/Mary/Urs')
├── Node('/Mary/Chris')
├── Node('/Mary/Marta')
└── Node('/Mary/Udo')
    ├── Node('/Mary/Udo/Marc')
    │   └── Node('/Mary/Udo/Marc/Lian')
    └── Node('/Mary/Udo/Dan')
        ├── Node('/Mary/Udo/Dan/Jet')
        ├── Node('/Mary/Udo/Dan/Jan')
        └── Node('/Mary/Udo/Dan/Joe')
```

Subtree rendering:

```
>>> print(RenderTree(marc))
Node('/Mary/Udo/Marc')
└── Node('/Mary/Udo/Marc/Lian')
```

Cut:

```
>>> dan.parent = None
>>> print(RenderTree(dan))
Node('/Dan')
├── Node('/Dan/Jet')
├── Node('/Dan/Jan')
└── Node('/Dan/Joe')
```

Extending any python class to become a tree node

The enitre tree magic is encapsulated by `NodeMixin`, add it as base class and the class becomes a tree node:

v: latest

```
>>> from anytree import NodeMixin, RenderTree
>>> class MyBaseClass(object):  # Just an example of a base class
...     foo = 4
```

```
>>> class MyClass(MyBaseClass, NodeMixin):  # Add Node feature
...     def __init__(self, name, length, width, parent=None, children=None):
...         super(MyClass, self).__init__()
...         self.name = name
...         self.length = length
...         self.width = width
...         self.parent = parent
...         if children:  # set children only if given
...             self.children = children
```

Just set the *parent* attribute to reflect the tree relation:

```
>>> my0 = MyClass('my0', 0, 0)
>>> my1 = MyClass('my1', 1, 0, parent=my0)
>>> my2 = MyClass('my2', 0, 2, parent=my0)
```

```
>>> for pre, fill, node in RenderTree(my0):
...     treestr = u"%s%s" % (pre, node.name)
...     print(treestr.ljust(8), node.length, node.width)
my0      0 0
├── my1  1 0
└── my2  0 2
```

The *children* can be used likewise:

```
>>> my0 = MyClass('my0', 0, 0, children=[
...     MyClass('my1', 1, 0),
...     MyClass('my2', 0, 2),
... ])
```

```
>>> for pre, fill, node in RenderTree(my0):
...     treestr = u"%s%s" % (pre, node.name)
...     print(treestr.ljust(8), node.length, node.width)
my0      0 0
├── my1  1 0
└── my2  0 2
```

v: latest