printree

Release 0.1.0

Christian López Barrón

Sep 17, 2020

CONTENTS:

Tree-like formatting for arbitrary python data structures.

CHAPTER

ONE

INSTALATION

```
pip install printree
```

CHAPTER

TWO

USAGE

`printree` aims to be similar to pretty print (pprint) with a format inspired by the tree command:

```
>>> from printree import ptree, ftree
>>> ptree({"x", len, 42})  # will print to the output console

├── 0: x
├── 1: <built-in function len>
└── 2: 42
>>> ftree({"x", len, 42})  # will return a string representation
'\n├── 0: x\n├── 1: <built-in function len>\n└── 2: 42'
```

Instances of abc.Iterable (with the exception of str & bytes) will be represented as branches. All other objects will be considered leaf nodes:

```
>>> from printree import ptree
>>> dct = {
...     "foo": [],
...     True: {
...         "uno": {"ABC", "XYZ"},
...         "dos": r"B:\newline\tab\like.ext",
...         "tres": {
...             "leaf": b"bytes",
...             "numbers": (42, -17, 0.01)
...         },
...     },
...     ("tuple", "as", "key"):
...         {"multi\nlined\n\ttabbed key": "multi\nline\n\ttabbed value"}
... }
>>> dct["recursion"] = [1, dct, 2]
>>> ptree(dct)

├── foo
├── True
│   ├── uno
│   │   ├── 0: XYZ
│   │   └── 1: ABC
│   ├── dos: B:\newline\tab\like.ext
│   └── tres
│       ├── leaf: b'bytes'
│       └── numbers
│           ├── 0: 42
│           ├── 1: -17
│           └── 2: 0.01
├── ('tuple', 'as', 'key')
```

(continues on next page)

(continued from previous page)

```
│   └── multi
    lined
            tabbed key: multi
                        line
                            tabbed value
└── recursion
    ├── 0: 1
    ├── 1: <Recursion on dict with id=2414949505984>
    └── 2: 2
```

The `annotated` and `depth` arguments modify verbosity of the output when creating the tree representation:

```
>>> ptree(dct, depth=2, annotated=True)
 → dict[items=4]
├── foo → list[empty]
├── True → dict[items=3]
│   ├── uno → set[items=2] [...]
│   ├── dos: B:\newline\tab\like.ext
│   └── tres → dict[items=2] [...]
├── ('tuple', 'as', 'key') → dict[items=1]
│   └── multi
    lined
            tabbed key: multi
                        line
                            tabbed value
└── recursion → list[items=3]
    ├── 0: 1
    ├── 1: <Recursion on dict with id=2414949505984>
    └── 2: 2
```

CHAPTER

THREE

CUSTOMIZING FORMATTING

`TreePrinter` subclasses can change each of the string representations of the tree. The subclass `AsciiPrinter` is provided as an example:

```
>>> from printree import AsciiPrinter
>>> obj = [42, {"foo": (True, False)}]
>>> AsciiPrinter(annotated=True).ptree(obj)
. -> list[items=2]
|-- 0: 42
`-- 1 -> dict[items=1]
    `-- foo -> tuple[items=2]
        |-- 0: True
        `-- 1: False
```

The main members to override are:

- `ROOT`
- `EDGE`
- `BRANCH_NEXT`
- `BRANCH_LAST`
- `ARROW`

The `level` attribute will be automatically set on the printer instance to indicate the current depth in the traversal of the tree.

To print each branch level with a different color, something like the following could be implemented:

```
from printree import TreePrinter

class ColoredTree(TreePrinter):
    colors = {
        0: '\033[31m',  # red
        1: '\033[32m',  # green
        2: '\033[33m',  # yellow
        3: '\033[36m',  # cyan
        4: '\033[35m',  # magenta
    }
    _RESET = '\033[0m'

    def __getattribute__(self, item):
        if item in ("EDGE", "BRANCH_NEXT", "BRANCH_LAST"):
            return f"{self.color}{getattr(super(), item)}{self._RESET}"
        return super().__getattribute__(item)
```

(continues on next page)

(continued from previous page)

```
    @property
    def color(self):
        return self.colors[self.level % len(self.colors)]

    @property
    def ROOT(self):  # for root (level 0), prefer the color of the children (level 1)
        return f'{self.colors[1]}{super().ROOT}{self._RESET}'


multiline = {"foo": {False: {"AB\nCD": "xy", 42:len}, True: []}, ("bar",): []}
dct = {"A": multiline, "B": (multiline,), "C\nD": "x\ny", "F": (1, "2")}

import os
os.system("")  # required on windows only

ColoredTree().ptree(dct)
```

Which outputs:

CHAPTER

FOUR

MODULE CONTENTS

class printree.**AsciiPrinter**(*depth=None*, *annotated=False*)
A printer that uses ASCII characters only.

class printree.**TreePrinter**(*depth=None*, *annotated=False*)
Default printer for printree.

Uses unicode characters.

property depth
Maximum depth to traverse while creating the tree representation.

Return type int

printree.**ftree**(*obj*, *depth=None*, *annotated=False*)
Return the formatted tree representation of the given object data structure as a string. Arguments are same as *ptree*

Return type str

printree.**ptree**(*obj*, *depth=None*, *annotated=False*)
Print a tree-like representation of the given object data structure.

collections.abc.Iterable instances will be branches, with the exception of str and bytes. All other objects will be leaves.

Parameters

- **depth** (Optional[int]) – If the data structure being printed is too deep, the next contained level is replaced by [...]. By default, there is no constraint on the depth of the objects being formatted.
- **annotated** (bool) – Whether or not to include annotations for branches, like the object type and amount of children.

Examples:

```
>>> dct = {"A": {"x\ny", (42, -17, 0.01), True}, "B": 42}
>>> ptree(dct)

├─ A
│  ├─ 0: x
│  │     y
│  ├─ 1
│  │  ├─ 0: 42
│  │  ├─ 1: -17
│  │  └─ 2: 0.01
```

(continues on next page)

(continued from previous page)

```
│   └─ 2: True
└─ B: 42
```

```
>>> ptree(dct, annotated=True, depth=2)
 → dict[items=2]
├─ A → set[items=3]
│   ├─ 0: x
│   │     y
│   ├─ 1 → tuple[items=3] [...]
│   └─ 2: True
└─ B: 42
```

Return type None

PYTHON MODULE INDEX

p

INDEX