pprint — Pretty-Print Data Structures

Purpose: Pretty-print data structures

The `pprint` module contains a “pretty printer” for producing aesthetically pleasing views of data structures. The formatter produces representations of data structures that can be parsed correctly by the interpreter, and that are also easy for a human to read. The output is kept on a single line, if possible, and indented when split across multiple lines.

The examples in this section all depend on `pprint_data.py`, which is shown here.

```
# pprint_data.py

data = [
    (1, {'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}),
    (2, {'e': 'E', 'f': 'F', 'g': 'G', 'h': 'H',
         'i': 'I', 'j': 'J', 'k': 'K', 'l': 'L'}),
    (3, ['m', 'n']),
    (4, ['o', 'p', 'q']),
    (5, ['r', 's', 't''u', 'v', 'x', 'y', 'z']),
]
```

Printing

The simplest way to use the module is through the `pprint()` function.

```
# pprint_pprint.py

from pprint import pprint

from pprint_data import data

print('PRINT:')
print(data)
print()
print('PPRINT:')
pprint(data)
```

`pprint()` formats an object and writes it to the data stream passed in as an argument (or `sys.stdout` by default).

```
$ python3 pprint_pprint.py

PRINT:
[(1, {'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}), (2, {'e': 'E', 'f':
'F', 'g': 'G', 'h': 'H', 'i': 'I', 'j': 'J', 'k': 'K', 'l': 'L'}), (
3, ['m', 'n']), (4, ['o', 'p', 'q']), (5, ['r', 's', 'tu', 'v', 'x',
 'y', 'z'])]

PPRINT:
[(1, {'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}),
 (2,
  {'e': 'E',
   'f': 'F',
   'g': 'G',
   'h': 'H',
   'i': 'I',
   'j': 'J',
   'k': 'K',
   'l': 'L'}),
 (3, ['m', 'n']),
 (4, ['o', 'p', 'q']),
 (5, ['r', 's', 'tu', 'v', 'x', 'y', 'z'])]
```

Formatting

To format a data structure without writing it directly to a stream (for example, for logging), use `pformat()` to build a string representation.

```
# pprint_pformat.py

import logging
from pprint import pformat
from pprint_data import data

logging.basicConfig(
    level=logging.DEBUG,
    format='%(levelname)-8s %(message)s',
)

logging.debug('Logging pformatted data')
formatted = pformat(data)
for line in formatted.splitlines():
    logging.debug(line.rstrip())
```

The formatted string can then be printed or logged independently.

```
$ python3 pprint_pformat.py

DEBUG    Logging pformatted data
DEBUG    [(1, {'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}),
DEBUG     (2,
DEBUG      {'e': 'E',
DEBUG       'f': 'F',
DEBUG       'g': 'G',
DEBUG       'h': 'H',
DEBUG       'i': 'I',
DEBUG       'j': 'J',
DEBUG       'k': 'K',
DEBUG       'l': 'L'}),
DEBUG     (3, ['m', 'n']),
DEBUG     (4, ['o', 'p', 'q']),
DEBUG     (5, ['r', 's', 'tu', 'v', 'x', 'y', 'z'])]
```

Arbitrary Classes

The `PrettyPrinter` class used by `pprint()` can also work with custom classes, if they define a `__repr__()` method.

```
# pprint_arbitrary_object.py
```

```
from pprint import pprint


class node:

    def __init__(self, name, contents=[]):
        self.name = name
        self.contents = contents[:]

    def __repr__(self):
        return (
            'node(' + repr(self.name) + ', ' +
            repr(self.contents) + ')'
        )


trees = [
    node('node-1'),
    node('node-2', [node('node-2-1')]),
    node('node-3', [node('node-3-1')]),
]
pprint(trees)
```

The representations of the nested objects are combined by the `PrettyPrinter` to return the full string representation.

```
$ python3 pprint_arbitrary_object.py

[node('node-1', []),
 node('node-2', [node('node-2-1', [])]),
 node('node-3', [node('node-3-1', [])])]
```

Recursion

Recursive data structures are represented with a reference to the original source of the data, given in the format `<Recursion on typename with id=number>`.

```
# pprint_recursion.py

from pprint import pprint

local_data = ['a', 'b', 1, 2]
local_data.append(local_data)

print('id(local_data) =>', id(local_data))
pprint(local_data)
```

In this example, the list `local_data` is added to itself, creating a recursive reference.

```
$ python3 pprint_recursion.py

id(local_data) => 4358913288
['a', 'b', 1, 2, <Recursion on list with id=4358913288>]
```

Limiting Nested Output

For very deep data structures, it may not be desirable for the output to include all of the details. The data may not be formatted properly, the formatted text might be too large to manage, or some of the data may be extraneous.

```
# pprint_depth.py

from pprint import pprint

from pprint_data import data

pprint(data, depth=1)
pprint(data, depth=2)
```

Use the `depth` argument to control how far down into the nested data structure the pretty printer recurses. Levels not included in the output are represented by ellipsis.

```
$ python3 pprint_depth.py

[(...), (...), (...), (...), (...)]
[(1, {...}), (2, {...}), (3, [...]), (4, [...]), (5, [...])]
```

Controlling Output Width

The default output width for the formatted text is 80 columns. To adjust that width, use the `width` argument to `pprint()`.

```
# pprint_width.py

from pprint import pprint

from pprint_data import data

for width in [80, 5]:
    print('WIDTH =', width)
    pprint(data, width=width)
```

```
    print()
```

When the width is too small to accommodate the formatted data structure, the lines are not truncated or wrapped if doing so would introduce invalid syntax.

```
$ python3 pprint_width.py

WIDTH = 80
[(1, {'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}),
 (2,
  {'e': 'E',
   'f': 'F',
   'g': 'G',
   'h': 'H',
   'i': 'I',
   'j': 'J',
   'k': 'K',
   'l': 'L'}),
 (3, ['m', 'n']),
 (4, ['o', 'p', 'q']),
 (5, ['r', 's', 'tu', 'v', 'x', 'y', 'z'])]

WIDTH = 5
[(1,
  {'a': 'A',
   'b': 'B',
   'c': 'C',
   'd': 'D'}),
 (2,
  {'e': 'E',
   'f': 'F',
   'g': 'G',
   'h': 'H',
   'i': 'I',
   'j': 'J',
   'k': 'K',
   'l': 'L'}),
 (3,
  ['m',
   'n']),
 (4,
  ['o',
   'p',
   'q']),
 (5,
  ['r',
   's',
   'tu',
   'v',
   'x',
   'y',
   'z'])]
```

The `compact` flag tells `pprint()` to try to fit more data on each individual line, rather than spreading complex data structures across lines.

```
# pprint_compact.py

from pprint import pprint

from pprint_data import data

print('DEFAULT:')
pprint(data, compact=False)
print('\nCOMPACT:')
pprint(data, compact=True)
```

This example shows that when a data structure does not fit on a line, it is split up (as with the second item in the data list). When multiple elements can fit on a line, as with the third and fourth members, they are placed that way.

```
$ python3 pprint_compact.py

DEFAULT:
[(1, {'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}),
 (2,
  {'e': 'E',
   'f': 'F',
   'g': 'G',
   'h': 'H',
   'i': 'I',
   'j': 'J',
   'k': 'K',
   'l': 'L'}),
 (3, ['m', 'n']),
 (4, ['o', 'p', 'q']),
 (5, ['r', 's', 'tu', 'v', 'x', 'y', 'z'])]

COMPACT:
[(1, {'a': 'A', 'b': 'B', 'c': 'C', 'd': 'D'}),
 (2,
  {'e': 'E',
   'f': 'F',
   'g': 'G',
   'h': 'H',
   'i': 'I',
   'j': 'J',
   'k': 'K',
   'l': 'L'}),
 (3, ['m', 'n']), (4, ['o', 'p', 'q']),
 (5, ['r', 's', 'tu', 'v', 'x', 'y', 'z'])]
```

See also

- Standard library documentation for pprint

copy — Duplicate Objects

Algorithms

Quick Links
Printing
Formatting
Arbitrary Classes
Recursion
Limiting Nested Output
Controlling Output Width

This page was last updated 2018-03-18.

Navigation
copy — Duplicate Objects
Algorithms

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example