pyclbr — Class Browser

Purpose: Implements an API suitable for use in a source code editor for making a class browser.

`pyclbr` can scan Python source to find classes and stand-alone functions. The information about class, method, and function names and line numbers is gathered using `tokenize` *without* importing the code.

The examples in this section use the following source file as input.

```python
# pyclbr_example.py

"""Example source for pyclbr.
"""


class Base:
    """This is the base class.
    """

    def method1(self):
        return


class Sub1(Base):
    """This is the first subclass.
    """


class Sub2(Base):
    """This is the second subclass.
    """


class Mixin:
    """A mixin class.
    """

    def method2(self):
        return


class MixinUser(Sub2, Mixin):
    """Overrides method1 and method2
    """

    def method1(self):
        return

    def method2(self):
        return

    def method3(self):
        return


def my_function():
    """Stand-alone function.
    """
    return
```

Scanning for Classes

There are two public functions exposed by `pyclbr`. The first, `readmodule()`, takes the name of the module as argument returns a mapping of class names to `Class` objects containing the metadata about the class source.

returns a mapping of class names to class objects containing the metadata about the class source.

```
# pyclbr_readmodule.py

import pyclbr
import os
from operator import itemgetter


def show_class(name, class_data):
    print('Class:', name)
    filename = os.path.basename(class_data.file)
    print('  File: {0} [{1}]'.format(
        filename, class_data.lineno))
    show_super_classes(name, class_data)
    show_methods(name, class_data)
    print()


def show_methods(class_name, class_data):
    for name, lineno in sorted(class_data.methods.items(),
                               key=itemgetter(1)):
        print('  Method: {0} [{1}]'.format(name, lineno))


def show_super_classes(name, class_data):
    super_class_names = []
    for super_class in class_data.super:
        if super_class == 'object':
            continue
        if isinstance(super_class, str):
            super_class_names.append(super_class)
        else:
            super_class_names.append(super_class.name)
    if super_class_names:
        print('  Super classes:', super_class_names)


example_data = pyclbr.readmodule('pyclbr_example')

for name, class_data in sorted(example_data.items(),
                               key=lambda x: x[1].lineno):
    show_class(name, class_data)
```

The metadata for the class includes the file and line number where it is defined, as well as the names of super classes. The methods of the class are saved as a mapping between method name and line number. The output shows the classes and methods listed in order based on their line number in the source file.

```
$ python3 pyclbr_readmodule.py

Class: Base
  File: pyclbr_example.py [11]
  Method: method1 [15]

Class: Sub1
  File: pyclbr_example.py [19]
  Super classes: ['Base']

Class: Sub2
  File: pyclbr_example.py [24]
  Super classes: ['Base']

Class: Mixin
  File: pyclbr_example.py [29]
  Method: method2 [33]

Class: MixinUser
  File: pyclbr_example.py [37]
  Super classes: ['Sub2', 'Mixin']
  Method: method1 [41]
  Method: method2 [44]
```

```
Method: method3 [47]
```

Scanning for Functions

The other public function in pyclbr is readmodule_ex(). It does everything that readmodule() does, and adds functions to the result set.

```
# pyclbr_readmodule_ex.py

import pyclbr
import os
from operator import itemgetter

example_data = pyclbr.readmodule_ex('pyclbr_example')

for name, data in sorted(example_data.items(),
                         key=lambda x: x[1].lineno):
    if isinstance(data, pyclbr.Function):
        print('Function: {0} [{1}]'.format(name, data.lineno))
```

Each Function object has properties much like the Class object.

```
$ python3 pyclbr_readmodule_ex.py

Function: my_function [51]
```

See also

- Standard library documentation for pyclbr
- inspect – The inspect module can discover more metadata about classes and functions, but requires importing the code.
- tokenize – The tokenize module parses Python source code into tokens.

compileall — Byte-compile Source Files

venv — Create Virtual Environments

This page was last updated 2016-12-31.

Navigation

compileall — Byte-compile Source Files
venv — Create Virtual Environments

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?