astor

Release 0.6

May 17, 2017

Contents

PyPI https://pypi.python.org/pypi/astor

Source https://github.com/berkerpeksag/astor

Issues https://github.com/berkerpeksag/astor/issues/

License 3-clause BSD

Build status

astor is designed to allow easy manipulation of Python source via the AST.

CHAPTER 1

Getting Started

Install with **pip**:

```
$ pip install astor
```

or clone the latest version from GitHub.

CHAPTER 2

Features

There are some other similar libraries, but astor focuses on the following areas:

- Round-trip back to Python via Armin Ronacher's codegen.py module:
  - Modified AST doesn't need linenumbers, ctx, etc. or otherwise be directly compileable
  - Easy to read generated code as, well, code
- Dump pretty-printing of AST
  - Harder to read than round-tripped code, but more accurate to figure out what is going on.
  - Easier to read than dump from built-in AST module
- Non-recursive treewalk
  - Sometimes you want a recursive treewalk (and astor supports that, starting at any node on the tree), but sometimes you don't need to do that. astor doesn't require you to explicitly visit sub-nodes unless you want to:
  - You can add code that executes before a node's children are visited, and/or
  - You can add code that executes after a node's children are visited, and/or
  - You can add code that executes and keeps the node's children from being visited (and optionally visit them yourself via a recursive call)
  - Write functions to access the tree based on object names and/or attribute names
  - Enjoy easy access to parent node(s) for tree rewriting

CHAPTER 3

Classes

class astor.**CodeToAst**
: This is the base class for the helper function code_to_ast. It may be subclassed, but probably will not need to be.

class astor.**TreeWalk**(*node=None*)
: The TreeWalk class is designed to be subclassed in order to walk a tree in arbitrary fashion. TreeWalk deserves its own treatise, but there is no time to write it at present :(

class astor.**ExplicitNodeVisitor**
: The ExplicitNodeVisitor class subclasses the ast.NodeVisitor class, and removes the ability to perform implicit visits. This allows for rapid failure when your code encounters a tree with a node type it was not expecting.

CHAPTER 4

Functions

Note: This section is not done. Please look at the source code for all public members.

astor.**to_source**(*source*, *indent_with=' ' * 4*, *add_line_information=False*)
Convert a node tree back into Python source code.

Each level of indentation is replaced with *indent_with*. Per default this parameter is equal to four spaces as suggested by **PEP 8**.

If *add_line_information* is set to `True` comments for the line numbers of the nodes are added to the output. This can be used to spot wrong line number information of statement nodes.

astor.**code_to_ast**(*codeobj*)
Given a module, or a function that was compiled as part of a module, re-compile the module into an AST and extract the sub-AST for the function. Allow caching to reduce number of compiles.

New in version 0.6: Was previously named codetoast

astor.code_to_ast.**parse_file**(*fname*)
Parse a python file into an AST.

This is a very thin wrapper around ast.parse

It does not yet handle all possible Python source encodings (issue #26).

New in version 0.6: Was previously named parsefile.

astor.code_to_ast.**get_file_info**(*codeobj*)
Returns the file and line number of a code object.

If the code object has a __file__ attribute (e.g. if it is a module), then the returned line number will be 0.

New in version 0.6.

astor.code_to_ast.**find_py_files**(*srctree*, *ignore=None*)
Recursively returns the path and filename for all .py files under the srctree directory.

If ignore is not None, it will ignore any path that contains the ignore string.

New in version 0.6.

astor.**iter_node**(*node*, *unknown=None*)

This function iterates over an AST node object:

- If the object has a _fields attribute, it gets attributes in the order of this and returns name, value pairs.
- Otherwise, if the object is a list instance, it returns name, value pairs for each item in the list, where the name is passed into this function (defaults to blank).
- Can update an unknown set with information about attributes that do not exist in fields.

astor.**dump_tree**(*node*, *name=None*, *initial_indent='*, *indentation=' '*, *maxline=120*, *maxmerged=80*)

This function pretty prints an AST or similar structure with indentation.

New in version 0.6: Was previously named dump

astor.**strip_tree**(*node*)

This function recursively removes all attributes from an AST tree that are not referenced by the _fields member.

Returns a set of the names of all attributes stripped. By default, this should just be the line number and column.

This canonicalizes two trees for comparison purposes.

New in version 0.6.

astor.**get_op_symbol**(*node*, *fmt='%s'*)

Given an ast node, returns the string representing the corresponding symbol.

New in version 0.6: Replaces and deprecates get_boolop, get_binop, get_cmpop, get_unaryop, and get_anyop.

CHAPTER 5

Command line utilities

rtrip

There is currently one command-line utility:

```
python -m astor.rtrip [readonly] [<source>]
```

This utility tests round-tripping of Python source to AST and back to source.

New in version 0.6.

If readonly is specified, then the source will be tested, but no files will be written.

if the source is specified to be "stdin" (without quotes) then any source entered at the command line will be compiled into an AST, converted back to text, and then compiled to an AST again, and the results will be displayed to stdout.

If neither readonly nor stdin is specified, then rtrip will create a mirror directory named tmp_rtrip and will recursively round-trip all the Python source from the source into the tmp_rtrip dir, after compiling it and then reconstituting it through code_gen.to_source.

If the source is not specified, the entire Python library will be used.

The purpose of rtrip is to place Python code into a canonical form.

This is useful both for functional testing of astor, and for validating code edits.

For example, if you make manual edits for PEP8 compliance, you can diff the rtrip output of the original code against the rtrip output of the edited code, to insure that you didn't make any functional changes.

For testing astor itself, it is useful to point to a big codebase, e.g:

```
python -m astor.rtrip
```

to round-trip the standard library.

If any round-tripped files fail to be built or to match, the tmp_rtrip directory will also contain fname.srcdmp and fname.dstdmp, which are textual representations of the ASTs.

Note 1: The canonical form is only canonical for a given version of this module and the astor toolbox. It is not guaranteed to be stable. The only desired guarantee is that two source modules that parse to the same AST will be converted back into the same canonical form.

Note 2: This tool WILL TRASH the tmp_rtrip directory (unless readonly is specified) – as far as it is concerned, it OWNS that directory.

CHAPTER 6

Release Notes

CHAPTER 7

0.6 - Work in Progress

New features

- New `astor.rtrip` submodule and doc
- New pretty printer outputs much better looking code
- Significant codebase refactoring - Much easier to add new node types - Much easier to handle precedence issues - Main flow easier to follow in tree
- Additional Python 3.5 support: - New starargs and kwargs support - Async and await
- Added Python 3.6 feature support:
  - f-strings
  - async comprehensions
  - variable annotations
- Updated test infrastructure and documentation
- Code cleanup, including renaming for PEP8 and deprecation of old names

Bug fixes

- TODO: Check issues...

CHAPTER 8

0.5 - 2015-04-18

New features

- Added support for Python 3.5 infix matrix multiplication

CHAPTER 9

0.4.1 - 2015-03-15

Bug fixes

- Added missing `SourceGenerator.visit_arguments()`

CHAPTER 10

0.4 - 2014-06-29

New features

- Added initial test suite and documentation

Bug fixes

- Added a visitor for `NameConstant`

CHAPTER 11

0.3 - 2013-12-10

New features

- Added support for Python 3.3.
  - Added `YieldFrom`
  - Updated `Try` and `With`.

Bug fixes

- Fixed a packaging bug on Python 3 – see pull requests #1 and #2 for more information.

CHAPTER 12

0.2.1 – 2012-09-20

Enhancements

- Modified TreeWalk to add `_name` suffix for functions that work on attribute names

CHAPTER 13

0.2 – 2012-09-19

Enhancements

- Initial Python 3 support
- Test of treewalk

CHAPTER 14

0.1 – 2012-09-19

- Initial release
- Based on Armin Ronacher's codegen
- Several bug fixes to that and new tree walker

Index