astor

Release 0.8.1

Dec 10, 2019

Contents

1 Release Notes **3**
- 1.1 0.8.1 - 2019-12-10 . . . 3
- 1.2 0.8.0 - 2019-05-19 . . . 3
- 1.3 0.7.1 - 2018-07-06 . . . 4
- 1.4 0.7.0 - 2018-07-05 . . . 4
- 1.5 0.6.2 - 2017-11-11 . . . 5
- 1.6 0.6.1 - 2017-11-11 . . . 5
- 1.7 0.6 - 2017-10-31 . . . 5
- 1.8 0.5 - 2015-04-18 . . . 6
- 1.9 0.4.1 - 2015-03-15 . . . 7
- 1.10 0.4 - 2014-06-29 . . . 7
- 1.11 0.3 - 2013-12-10 . . . 7
- 1.12 0.2.1 – 2012-09-20 . . . 7
- 1.13 0.2 – 2012-09-19 . . . 7
- 1.14 0.1 – 2012-09-19 . . . 8

2 Getting Started **9**

3 Features **11**

4 Deprecations **13**
- 4.1 Modules . . . 13
- 4.2 Functions . . . 13
- 4.3 Attributes . . . 14

5 Functions **15**

6 Classes **17**

7 Command-line utilities **19**
- 7.1 rtrip . . . 19

Index **21**

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

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

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

License 3-clause BSD

Build status

CHAPTER 1

Release Notes

1.1 0.8.1 - 2019-12-10

1.1.1 Bug fixes

- Fixed precedence issue for f-string expressions that caused redundant parenthesis around expression. (Reported by Ilya Kamenshchikov in Issue 153 and fixed by Batuhan Taskaya in PR 155.)
- Fixed `astor.to_source()` incorrectly checking whether *source_generator_class* is a subclass of `astor.code_gen.SourceGenerator`. (Reported by Yu-Chia "Hank" Liu in Issue 158 and fixed by Will Crichton in PR 164.)
- Fixed `TypeError` when AST nodes with unicode strings are passed to `astor.to_source()`. (Reported and fixed by Dominik Moritz in PR 154.)
- Fixed installation issue with setuptools 41.4.0 or later due to the use of an undocumented feature. (Reported and fixed by Jonathan Ringer in Issue 162 and PR 163.)

1.2 0.8.0 - 2019-05-19

1.2.1 New features

- Support `ast.Constant` nodes being emitted by Python 3.8 (and initially created in Python 3.6). (Reported and fixed by Chris Rink in Issue 120 and PR 121.)
- Support Python 3.8's assignment expressions. (Reported and fixed by Kodi Arfer in Issue 126 and PR 134.)
- Support Python 3.8's f-string debugging syntax. (Reported and fixed by Batuhan Taskaya in Issue 138 and PR 139.)
- `astor.to_source()` now has a *source_generator_class* parameter to customize source code generation. (Reported and fixed by matham in Issue 113 and PR 114.)

- The `SourceGenerator` class can now be imported from the `astor` package directly. Previously, the `astor.code_gen` submodule was needed to be imported.
- Support Python 3.8's positional only arguments. See **PEP 570** for more details. (Reported and fixed by Batuhan Taskaya in Issue 142 and PR 143.)

1.2.2 Bug fixes

- Fix string parsing when there is a newline inside an f-string. (Reported by Adam Cécile in Issue 119 and fixed by Felix Yan in PR 123.)
- Fixed code generation with escaped braces in f-strings. (Reported by Felix Yan in Issue 124 and fixed by Kodi Arfer in PR 125.)
- Fixed code generation with attributes of integer literals, and with `u`-prefixed string literals. (Fixed by Kodi Arfer in PR 133.)
- Fixed code generation with very large integers. (Reported by Adam Kucz in Issue 127 and fixed by Kodi Arfer in PR 130.)
- Fixed *`astor.tree_walk.TreeWalk`* when attempting to access attributes created by Python's type system (such as `__dict__` and `__weakref__`) (Reported and fixed by esupoff in Issue 136 and PR 137.)

1.3 0.7.1 - 2018-07-06

1.3.1 Bug fixes

- Fixed installation error by adding the `setuputils.py` helper to the sdist. (Reported by Adam and fixed by Berker Peksag in Issue 116.)

1.4 0.7.0 - 2018-07-05

1.4.1 New features

- Added initial support for Python 3.7.0.

  Note that if you have a subclass of `astor.code_gen.SourceGenerator`, you may need to rename the keyword argument `async` of the following methods to `is_async`:

  - `visit_FunctionDef(..., is_async=False)`
  - `visit_For(..., is_async=False)`
  - `visit_With(..., is_async=False)`

  (Reported and fixed by Berker Peksag in Issue 86.)
- Dropped support for Python 2.6 and Python 3.3.

1.4.2 Bug fixes

- Fixed a bug where newlines would be inserted to a wrong place during printing f-strings with trailing newlines. (Reported by Felix Yan and contributed by Radomír Bosák in Issue 89.)

- Improved code generation to support `ast.Num` nodes containing infinities or NaNs. (Reported and fixed by Kodi Arfer in Issue 85 and Issue 100.)
- Improved code generation to support empty sets. (Reported and fixed by Kodi Arfer in Issue 108.)

1.5 0.6.2 - 2017-11-11

1.5.1 Bug fixes

- Restore backwards compatibility that was broken after 0.6.1. You can now continue to use the following pattern:

```
import astor

class SpamCodeGenerator(astor.codegen.SourceGenerator):
    ...
```

(Reported by Dan Moldovan and fixed by Berker Peksag in Issue 87.)

1.6 0.6.1 - 2017-11-11

1.6.1 New features

- Added `astor.parse_file()` as an alias to `astor.code_to_ast.parsefile()`. (Contributed by Berker Peksag.)

1.6.2 Bug fixes

- Fix compatibility layer for the `astor.codegen` submodule. Importing `astor.codegen` now succeeds and raises a `DeprecationWarning` instead of `ImportError`. (Contributed by Berker Peksag.)

1.7 0.6 - 2017-10-31

1.7.1 New features

- New `astor.rtrip` command-line tool to test round-tripping of Python source to AST and back to source. (Contributed by Patrick Maupin.)
- New pretty printer outputs much better looking code:
  - Remove parentheses where not necessary
  - Use triple-quoted strings where it makes sense
  - Add placeholder for function to do nice line wrapping on output

  (Contributed by Patrick Maupin.)
- Additional Python 3.5 support:
  - Additional unpacking generalizations (**PEP 448**)
  - Async and await (**PEP 492**)

(Contributed by Zack M. Davis.)

- Added Python 3.6 feature support:
  - f-strings (**PEP 498**)
  - async comprehensions (**PEP 530**)
  - variable annotations (**PEP 526**)

  (Contributed by Ryan Gonzalez.)
- Code cleanup, including renaming for PEP8 and deprecation of old names. See *Deprecations* for more information. (Contributed by Leonard Truong in Issue 36.)

1.7.2 Bug fixes

- Don't put trailing comma-spaces in dictionaries. astor will now create `{'three': 3}` instead of `{'three': 3, }`. (Contributed by Zack M. Davis.)
- Fixed several bugs in code generation:
  1. Keyword-only arguments should come before `**`
  2. `from .. import <member>` with no trailing module name did not work
  3. Support `from .. import foo as bar` syntax
  4. Support `with foo: ...`, `with foo as bar: ...` and `with foo, bar as baz: ...` syntax
  5. Support `1eNNNN` syntax
  6. Support `return (yield foo)` syntax
  7. Support unary operations such as `-(1) + ~(2) + +(3)`
  8. Support `if (yield): pass`
  9. Support `if (yield from foo): pass`
  10. `try...finally` block needs to come after the `try...else` clause
  11. Wrap integers with parentheses where applicable (e.g. `(0).real` should generated)
  12. When the `yield` keyword is an expression rather than a statement, it can be a syntax error if it is not enclosed in parentheses
  13. Remove extraneous parentheses around `yield from`

  (Contributed by Patrick Maupin in Issue 27.)

1.8 0.5 - 2015-04-18

1.8.1 New features

- Added support for Python 3.5 infix matrix multiplication (**PEP 465**) (Contributed by Zack M. Davis.)

1.9 0.4.1 - 2015-03-15

1.9.1 Bug fixes

- Added missing `SourceGenerator.visit_arguments()`

1.10 0.4 - 2014-06-29

1.10.1 New features

- Added initial test suite and documentation

1.10.2 Bug fixes

- Added a visitor for `NameConstant`

1.11 0.3 - 2013-12-10

1.11.1 New features

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

1.11.2 Bug fixes

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

1.12 0.2.1 – 2012-09-20

1.12.1 Enhancements

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

1.13 0.2 – 2012-09-19

1.13.1 Enhancements

- Initial Python 3 support
- Test of treewalk

1.14 0.1 – 2012-09-19

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

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

CHAPTER 2

Getting Started

Install with **pip**:

```
$ pip install astor
```

or clone the latest version from GitHub.

CHAPTER 3

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 4

Deprecations

New in version 0.6.

4.1 Modules

astor 0.5	astor 0.6+
`astor.codegen`	`astor.code_gen`
`astor.misc`	`astor.file_util`
`astor.treewalk`	`astor.tree_walk`

4.2 Functions

astor 0.5	astor 0.6+
`astor.codetoast()`	`astor.code_to_ast()`
`astor.parsefile()`	`astor.parse_file()`
`astor.dump()`	`astor.dump_tree()`
`astor.get_anyop()`	`astor.get_op_symbol()`
`astor.get_boolop()`	`astor.get_op_symbol()`
`astor.get_binop()`	`astor.get_op_symbol()`
`astor.get_cmpop()`	`astor.get_op_symbol()`
`astor.get_unaryop()`	`astor.get_op_symbol()`

4.3 Attributes

astor 0.5	astor 0.6+
`astor.codetoast`	`astor.code_to_ast`
`astor.all_symbols`	`astor.symbol_data`

CHAPTER 5

Functions

to_source(source, indent_with=' ' * 4, add_line_information=False, source_generator_class=astor.SourceGenerator)

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.

source_generator_class defaults to `astor.SourceGenerator`, and specifies the class that will be instantiated and used to generate the source code.

Changed in version 0.8: *source_generator_class* was added.

`astor.`**`codetoast`**`()`

`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.

Deprecated since version 0.6: `codetoast()` is deprecated.

`astor.`**`parsefile`**`()`

`astor.`**`parse_file`**`()`

`astor.code_to_ast.`**`parse_file`**`(`*fname*`)`

Parse a Python file into an AST.

This is a very thin wrapper around `ast.parse()`.

Deprecated since version 0.6: `astor.parsefile()` is deprecated.

New in version 0.6.1: Added the `astor.parse_file()` function as an alias.

`astor.code_to_ast.`**`get_file_info`**`(`*codeobj*`)`

Returns the file and line number of *codeobj*.

If *codeobj* 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 Python 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**()

astor.**dump_tree**(*node*, *name=None*, *initial_indent=''*, *indentation=' '*, *maxline=120*, *maxmerged=80*)
This function pretty prints an AST or similar structure with indentation.

Deprecated since version 0.6: astor.dump() is deprecated.

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_boolop**()

astor.**get_binop**()

astor.**get_cmpop**()

astor.**get_unaryop**()

astor.**get_anyop**()

astor.**get_op_symbol**(*node*, *fmt='%s'*)
Given an ast node, returns the string representing the corresponding symbol.

Deprecated since version 0.6: get_boolop(), get_binop(), get_cmpop(), get_unaryop() and get_anyop() functions are deprecated.

CHAPTER 6

Classes

class file_util.**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 tree_walk.**TreeWalk**(*node=None*)
: The TreeWalk class is designed to be subclassed in order to walk a tree in arbitrary fashion.

class node_util.**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 7

Command-line utilities

There is currently one command-line utility:

7.1 rtrip

New in version 0.6.

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

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

Warning: This tool **will trash** the *tmp_rtrip* directory unless the *readonly* option is specified.

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: 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.

Index

A

astor.code_to_ast.find_py_files() (*in module astor*), 16
astor.code_to_ast.get_file_info() (*in module astor*), 15
astor.code_to_ast.parse_file() (*in module astor*), 15
astor.parse_file() (*in module astor*), 15
astor.parsefile() (*in module astor*), 15

C

code_to_ast() (*in module astor*), 15
codetoast() (*in module astor*), 15

D

dump() (*in module astor*), 16
dump_tree() (*in module astor*), 16

F

file_util.CodeToAst (*class in astor*), 17

G

get_anyop() (*in module astor*), 16
get_binop() (*in module astor*), 16
get_boolop() (*in module astor*), 16
get_cmpop() (*in module astor*), 16
get_op_symbol() (*in module astor*), 16
get_unaryop() (*in module astor*), 16

I

iter_node() (*in module astor*), 16

N

node_util.ExplicitNodeVisitor (*class in astor*), 17

P

Python Enhancement Proposals
    PEP 448, 5
    PEP 465, 6
    PEP 492, 5
    PEP 498, 6
    PEP 526, 6
    PEP 530, 6
    PEP 570, 4
    PEP 8, 15

S

strip_tree() (*in module astor*), 16

T

tree_walk.TreeWalk (*class in astor*), 17