

---

# **astcheck Documentation**

*Release 0.2.4*

**Thomas Kluyver**

**Oct 17, 2021**



---

## Contents

---

<b>1</b>	<b>Checking ASTs</b>	<b>3</b>
1.1	Checker functions . . . . .	3
1.2	Exceptions . . . . .	4
<b>2</b>	<b>Template building utilities</b>	<b>5</b>
<b>3</b>	<b>Changes</b>	<b>7</b>
3.1	Version 0.3 . . . . .	7
3.2	Version 0.2 . . . . .	7
<b>4</b>	<b>Indices and tables</b>	<b>9</b>
	<b>Python Module Index</b>	<b>11</b>
	<b>Index</b>	<b>13</b>



astcheck is a module for checking a Python Abstract Syntax Tree against a template. This is useful for testing code that automatically generates or modifies Python code.

For more details about the structure of ASTs, see my documentation project, [Green Tree Snakes](#).



Start by defining a template, a partial AST to compare against. You can fill in as many or as few fields as you want. For example, to check for assignment to a variable `a`, but ignore the value:

```
template = ast.Module(body=[
    ast.Assign(targets=[ast.Name(id='a')])
])
sample = ast.parse("a = 7")
astcheck.assert_ast_like(sample, template)
```

`astcheck` provides some helpers for defining flexible templates; see *Template building utilities*.

`astcheck.assert_ast_like` (*sample*, *template*, *\_path=None*)

Check that the sample AST matches the template.

Raises a suitable subclass of *ASTMismatch* if a difference is detected.

The `_path` parameter is used for recursion; you shouldn't normally pass it.

`astcheck.is_ast_like` (*sample*, *template*)

Returns True if the sample AST matches the template.

---

**Note:** The parameter order matters! Only fields present in `template` will be checked, so you can leave out bits of the code you don't care about. Normally, `sample` will be the result of the code you want to test, and `template` will be defined in your test file.

---

## 1.1 Checker functions

You may want to write more customised checks for part of the AST. To do so, you can attach 'checker functions' to any part of the template tree. Checker functions should accept two parameters: the node or value at the corresponding part of the sample tree, and the path to that node—a list of strings and integers representing the attribute and index access used to get there from the root of the sample tree.

If the value passed is not acceptable, the checker function should raise one of the exceptions described below. Otherwise, it should return with no exception. The return value is ignored.

For instance, this will test for a number literal less than 7:

```
def less_than_seven(node, path):
    if not isinstance(node, ast.Num):
        raise astcheck.ASTNodeTypeMismatch(path, node, ast.Num())
    if node.n >= 7:
        raise astcheck.ASTMismatch(path+['n'], node.n, '< 7')

template = ast.Expression(body=ast.BinOp(left=less_than_seven))
sample = ast.parse('4+9', mode='eval')
astcheck.assert_ast_like(sample, template)
```

There are a few checker functions available in `astcheck`—see *Template building utilities*.

## 1.2 Exceptions

**exception** `astcheck.ASTMismatch` (*path, got, expected*)

Bases: `exceptions.AssertionError`

Base exception for differing ASTs.

The following exceptions are raised by `assert_ast_like()`. They should all produce useful error messages explaining which part of the AST differed and how:

**exception** `astcheck.ASTNodeTypeMismatch` (*path, got, expected*)

Bases: `astcheck.ASTMismatch`

An AST node was of the wrong type.

**exception** `astcheck.ASTNodeListMismatch` (*path, got, expected*)

Bases: `astcheck.ASTMismatch`

A list of AST nodes had the wrong length.

**exception** `astcheck.ASTPlainListMismatch` (*path, got, expected*)

Bases: `astcheck.ASTMismatch`

A list of non-AST objects did not match.

e.g. A `ast.Global` node has a `names` list of plain strings

**exception** `astcheck.ASTPlainObjMismatch` (*path, got, expected*)

Bases: `astcheck.ASTMismatch`

A single value, such as a variable name, did not match.



---

## Template building utilities

---

`astcheck` includes some utilities for building AST templates to check against.

`astcheck.must_exist` (*node*, *path*)

Checker function for an item or list that must exist

This matches any value except `None` and the empty list.

For instance, to match for loops with an else clause:

```
ast.For(orelse=astcheck.must_exist)
```

`astcheck.must_not_exist` (*node*, *path*)

Checker function for things that must not exist

This accepts only `None` and the empty list.

For instance, to check that a function has no decorators:

```
ast.FunctionDef(decorator_list=astcheck.must_not_exist)
```

**class** `astcheck.name_or_attr` (*name*)

Checker for `ast.Name` or `ast.Attribute`

These are often used in similar ways - depending on how you do imports, objects will be referenced as names or as attributes of a module. By using this function to build your template, you can allow either. For instance, this will match both `f()` and `mod.f()`:

```
ast.Call(func=astcheck.name_or_attr('f'))
```

**class** `astcheck.single_assign` (*target=None*, *value=None*)

Checker for `ast.Assign` or `ast.AnnAssign`

`Assign` is a plain assignment. This will match only assignments to a single target (`a = 1` but not `a = b = 1`). `AnnAssign` is an annotated assignment, like `a: int = 7`. *target* and *value* may be AST nodes to check, so this would match any assignment to `a`:

```
astcheck.single_assign(target=ast.Name(id='a'))
```

Annotated assignments don't necessarily have a value: `a: int` is parsed as an `AnnAssign` node. Use `must_exist()` to avoid matching these:

```
astcheck.single_assign(target=ast.Name(id='a'), value=astcheck.must_exist)
```

**class** `astcheck.listmiddle`

Helper to check only the beginning and/or end of a list. Instantiate it and add lists to it to match them at the start or the end. E.g. to test the final return statement of a function, while ignoring any other code in the function:

```
template = ast.FunctionDef(name="myfunc",
    body= astcheck.listmiddle()+[ast.Return(value=ast.Name(id="retval"))]
)

sample = ast.parse("""
def myfunc():
    retval = do_something() * 7
    return retval
""")

astcheck.assert_ast_like(sample.body[0], template)
```

### 3.1 Version 0.3

- Added *single\_assign* helper.
- Requires Python  $\geq 3.5$ .

### 3.2 Version 0.2

- Added *checker functions*.



## CHAPTER 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**a**

astcheck, 3





## A

`assert_ast_like()` (*in module astcheck*), 3  
`astcheck` (*module*), 3  
`ASTMismatch`, 4  
`ASTNodeListMismatch`, 4  
`ASTNodeTypeMismatch`, 4  
`ASTPlainListMismatch`, 4  
`ASTPlainObjMismatch`, 4

## I

`is_ast_like()` (*in module astcheck*), 3

## L

`listmiddle` (*class in astcheck*), 6

## M

`must_exist()` (*in module astcheck*), 5  
`must_not_exist()` (*in module astcheck*), 5

## N

`name_or_attr` (*class in astcheck*), 5

## S

`single_assign` (*class in astcheck*), 5