AIRFLOW SUMMIT

Seamless Migration

Leveraging Ruff for a Smooth Transition from Airflow 2 to Airflow 3

3.0

Wei Lee @ astronomer

Seamless Migration

Leveraging Ruff for a Smooth Transition from Airflow 2 to Airflow 3

🛝 Slide deck



What to expect? 👀

1. What is Ruff?
2. How Ruff helps Airflow Migration?
3. How is it done?

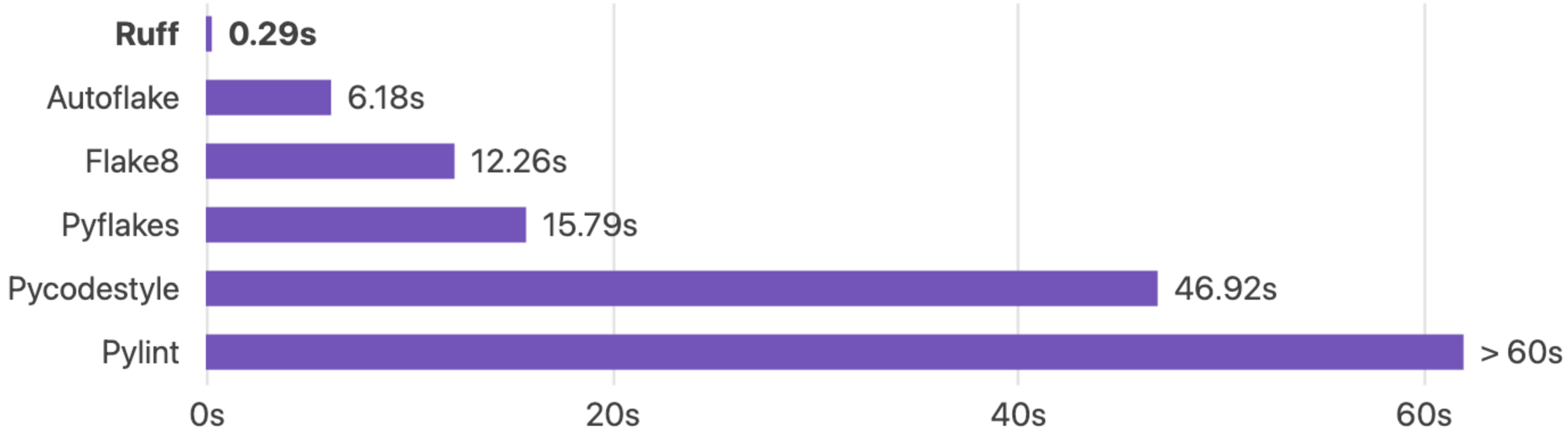
Ch 1 - What is Ruff

Ruff

Ruff | pypi v0.13.1 | license MIT | python 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12 | 3.13 | CI passing | Discord

Docs | Playground

An extremely fast Python linter and code formatter, written in Rust.



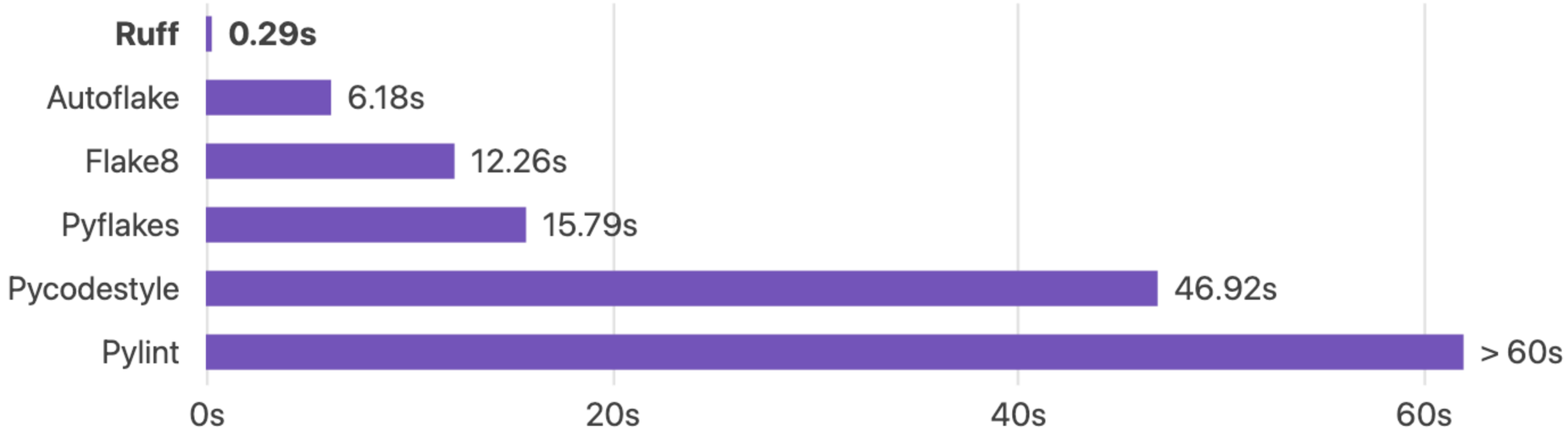
Linting the CPython codebase from scratch.

Ruff

Ruff | pypi v0.13.1 | license MIT | python 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12 | 3.13 | CI passing | Discord

Docs | Playground

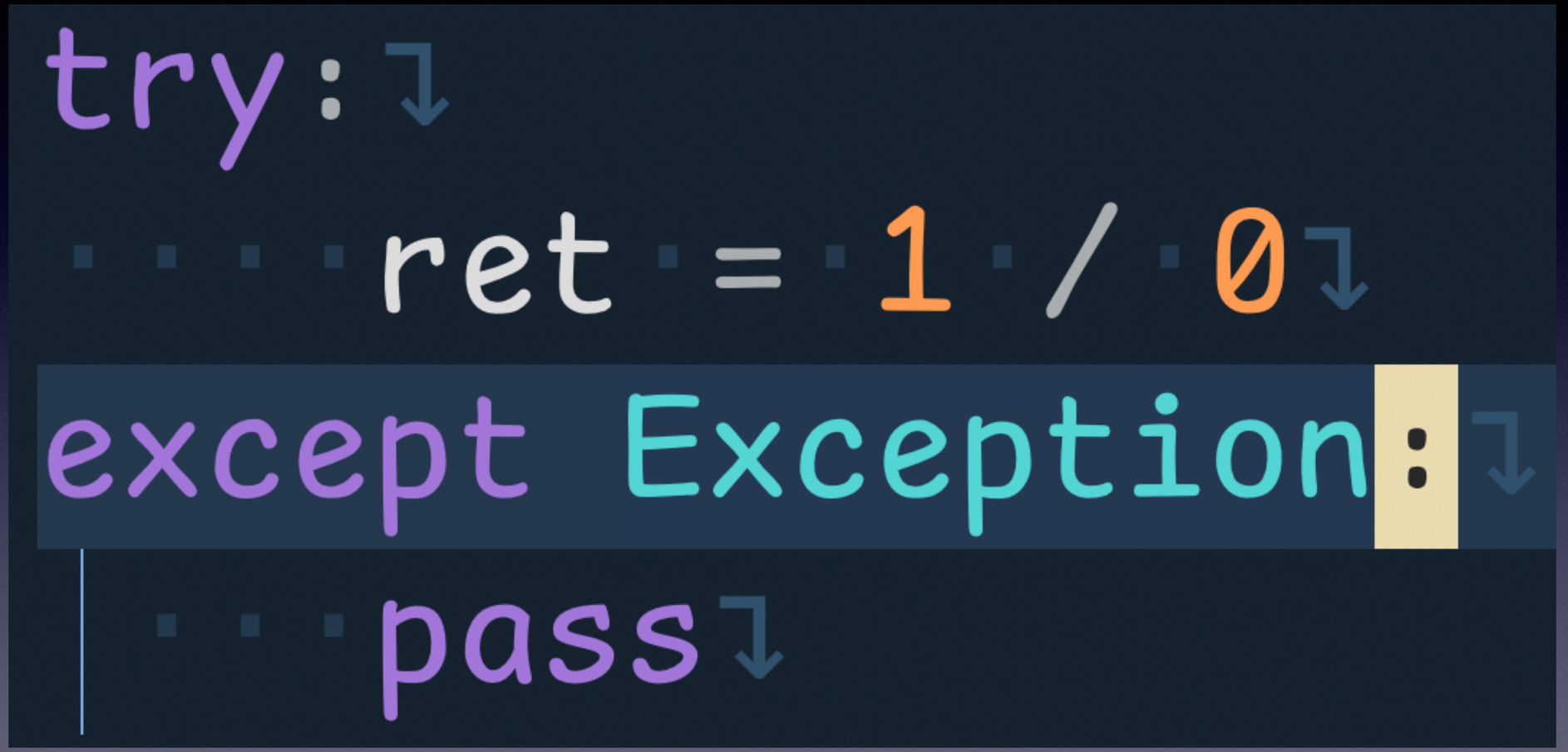
An extremely fast Python linter and code formatter, written in Rust.



Linting the CPython codebase from scratch.

Linter

- A static analysis tool that checks
  - Bad coding style
  - Potential error before running


try:
ret = 1 / 0
except Exception:
pass

```
✗  uv run ruff check example.py --select BLE001
BLE001 Do not catch blind exception: `Exception`
 --> example.py:3:8
  |
1 | try:
2 |     ret = 1 / 0
3 | except Exception:
  |        ^^^^^^^^^
  |
4 |     pass
  |


Found 1 error.
```

blind-except (BLE001)

Derived from the **flake8-blind-except** linter.

What it does

Checks for `except` clauses that catch all exceptions. This includes `except BaseException` and `except Exception`.

Why is this bad?

Overly broad `except` clauses can lead to unexpected behavior, such as catching `KeyboardInterrupt` or `SystemExit` exceptions that prevent the user from exiting the program.

Instead of catching all exceptions, catch only those that are expected to be raised in the `try` block.

It's as bad as raising AirflowException





Wei Lee - Wednesday, September 10, 2025 3:31:04 AM GMT+9

[LAZY CONSENSUS] Avoid adding new direct `AirflowException` usage

Hi all,

Following positive feedback on https://lists.apache.org/thread/5rv4tz0oc27bgr4khx0on0jz8fpxvh55, I propose a lazy consensus to avoid new direct *AirflowException* usage (i.e., *raise AirflowException*).

Relevant PRs:
* docs(contributing-docs): avoid raise AirflowException directly #55420 <https://github.com/apache/airflow/pull/55420>: adding this as the best practices in the contribution guide
* build(prek-hook): Check whether new "raise AirflowException" is added #55416 <https://github.com/apache/airflow/pull/55416>: adding a prek hook to block "raise AirflowException"

If there are no objections by Friday, the 12th of September 2025, 1 pm AoE, this will be considered accepted.

Best regards,
Wei Lee



Wei Lee - Saturday, September 13, 2025 3:55:00 PM GMT+9

Lazy consensus passed. Mark the 2 PRs as ready to review.

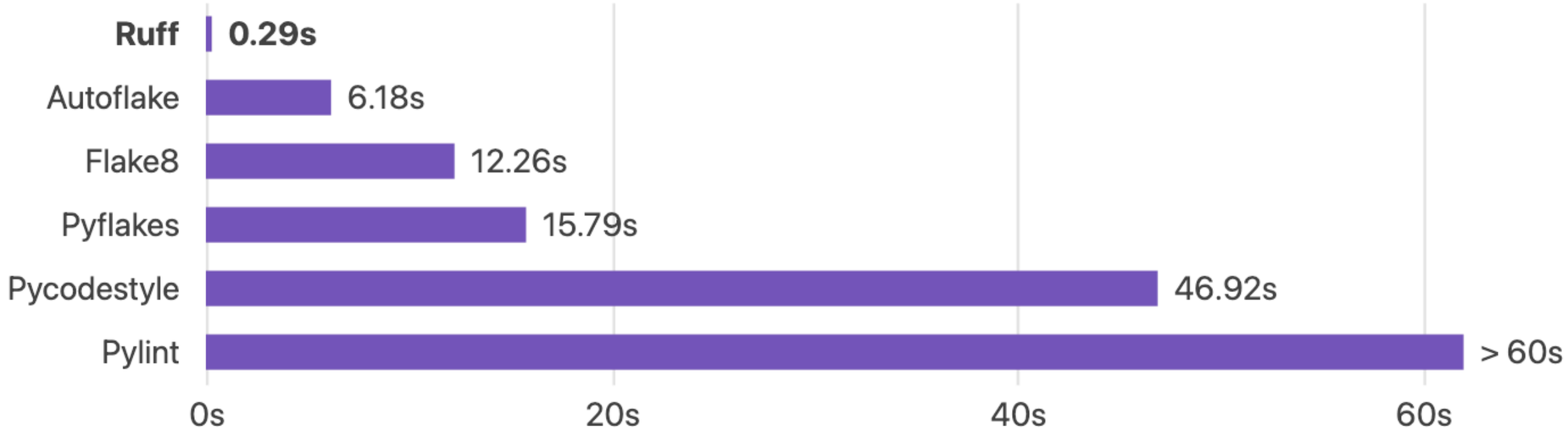
Wei Lee <we...@gmail.com> 於 2025年9月9日 週二 上午11:31寫道：

Ruff

Ruff | pypi v0.13.1 | license MIT | python 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12 | 3.13 | CI passing | Discord

Docs | Playground

An extremely fast Python linter and code formatter, written in Rust.



Linting the CPython codebase from scratch.

Before

```
def add( a:int,b:int )->int: return a+b
nums=[1,2,3,4,5]
s=sum( nums )
print( "Sum is",s )
```



After

```
def add(a: int, b: int) -> int:
    return a + b


nums = [1, 2, 3, 4, 5]
s = sum(nums)
print("Sum is", s)
```

Ch 2

How Ruff helps Airflow Migration



Iliya Rom... • 2nd ✓ Following ···

Research Software Archi...

2d •

If you're still using Airflow 2.x, now's a great time to upgrade (straight to 3.1) and enjoy all the new features and UI improvements, as well as Python 3.13 support! Upgrading is quite painless owing to all the automated tools such as Ruff inspections with auto-fixes (awesome work - **Wei Lee**!) and configuration upgrade scripts.

How I migrated a community Airflow instance to Airflow 3



With Ruff AIR rules

5 dags/app/proposal_reminder/dag.py

```
@@ -4,10 +4,11 @@
4  4
5  5  from datetime import datetime, timedelta
6  6
7     from airflow.decorators import dag, task
8     from airflow.models import Variable
9  7  from app import discord
10 8  from app.proposal_reminder.udf import get_proposal_summary
   9  from airflow.sdk import dag
   10 from airflow.sdk import task
   11 from airflow.sdk import Variable
```

```
→ uv run \
ruff check example.py \
--select AIR \
--fix \
--unsafe-fixes
```

Use Ruff >= 0.13.0

Stabilize the remaining Airflow rules #20250

Merged `brent/0.13.0` ← ~~`brent/airflow`~~ last month

Conversation 8 | Commits 5 | Checks 35 | Files changed 4



ntBre (Brent Westbrook) on Sep 5 • edited — Contributor

- Stabilize `airflow3-suggested-update` (`AIR311`)
- Stabilize `airflow3-suggested-to-move-to-provider` (`AIR312`)
- Stabilize `airflow3-removal` (`AIR301`)
- Stabilize `airflow3-moved-to-provider` (`AIR302`)
- Stabilize `airflow-dag-no-schedule-argument` (`AIR002`)

Run ruff with uv

```
→ uv run \
ruff check example.py \
--select AIR \
--fix \
--unsafe-fixes
```

Check example.py through Ruff

```
→ uv run \
ruff check example.py \
--select AIR \
--fix \
--unsafe-fixes
```

Select all AIR rules

```
→ uv run \
ruff check example.py \
--select AIR \
--fix \
--unsafe-fixes
```



Rules

Airflow (AIR)

For more, see Airflow on PyPI.

Fix

```
→ uv run \
ruff check example.py \
--select AIR \
--fix \
--unsafe-fixes
```

Ruff AIR rules

Airflow (AIR)

For more, see Airflow on PyPI.

Code	Name	Message	
AIR001	airflow-variable-name-task-id-mismatch	Task variable name should match the `task_id`: "{task_id}"	
AIR002	airflow-dag-no-schedule-argument	`DAG` or `@dag` should have an explicit `schedule` argument	
AIR301	airflow3-removal	`{deprecated}` is removed in Airflow 3.0	🛠
AIR302	airflow3-moved-to-provider	`{deprecated}` is moved into `{provider}` provider in Airflow 3.0;	🛠
AIR311	airflow3-suggested-update	`{deprecated}` is removed in Airflow 3.0; It still works in Airflow 3.0 but is expected to be removed in a future version.	🛠
AIR312	airflow3-suggested-to-move-to-provider	`{deprecated}` is deprecated and moved into `{provider}` provider in Airflow 3.0; It still works in Airflow 3.0 but is expected to be removed in a future version.	🛠

🛠️ means fixable

Airflow (AIR)

For more, see Airflow on PyPI.

Code	Name	Message	
AIR001	airflow-variable-name-task-id-mismatch	Task variable name should match the `task_id`: "{task_id}"	
AIR002	airflow-dag-no-schedule-argument	`DAG` or `@dag` should have an explicit `schedule` argument	
AIR301	airflow3-removal	`{deprecated}` is removed in Airflow 3.0	🛠️
AIR302	airflow3-moved-to-provider	`{deprecated}` is moved into `{provider}` provider in Airflow 3.0;	🛠️
AIR311	airflow3-suggested-update	`{deprecated}` is removed in Airflow 3.0; It still works in Airflow 3.0 but is expected to be removed in a future version.	🛠️
AIR312	airflow3-suggested-to-move-to-provider	`{deprecated}` is deprecated and moved into `{provider}` provider in Airflow 3.0; It still works in Airflow 3.0 but is expected to be removed in a future version.	🛠️

Include unsafe fixes

```
→ uv run \
ruff check example.py \
--select AIR \
--fix \
--unsafe-fixes
```

Unsafe fixes ?! 😱

Fix safety

Ruff labels fixes as "safe" and "unsafe". The meaning and intent of your code will be retained when applying safe fixes, but the meaning could change when applying unsafe fixes.

Specifically, an unsafe fix could lead to a change in runtime behavior, the removal of comments, or both, while safe fixes are intended to preserve runtime behavior and will only remove comments when deleting entire statements or expressions (e.g., removing unused imports).

Unsafe fixes ?! 😱

Fix safety

Ruff labels fixes as "safe" and "unsafe". The meaning and intent of your code will be retained when applying safe fixes, but the meaning could change when applying unsafe fixes.

Specifically, an unsafe fix could lead to a change in runtime behavior, the removal of comments, or both, while safe fixes are intended to preserve runtime behavior and will only remove comments when deleting entire statements or expressions (e.g., removing unused imports).

For AIR rules,

Unsafe fixes are mostly fine.



Ruff AIR rules

Code	Name	Message	
AIR001	airflow-variable-name-task-id-mismatch	Task variable name should match the `task_id`: "{task_id}"	
AIR002	airflow-dag-no-schedule-argument	`DAG` or `@dag` should have an explicit `schedule` argument	
AIR301	airflow3-removal	`{deprecated}` is removed in Airflow 3.0	🛠️
AIR302	airflow3-moved-to-provider	`{deprecated}` is moved into `{provider}` provider in Airflow 3.0;	🛠️
AIR311	airflow3-suggested-update	`{deprecated}` is removed in Airflow 3.0; It still works in Airflow 3.0 but is expected to be removed in a future version.	🛠️
AIR312	airflow3-suggested-to-move-to-provider	`{deprecated}` is deprecated and moved into `{provider}` provider in Airflow 3.0; It still works in Airflow 3.0 but is expected to be removed in a future version.	🛠️

AIR Rule Code Convention

(for the time being)

AIR X Y Z



Apply to Airflow X and onward

Ruff AIR Rules

Best practices 👍

- AIR001
  Task Variable name should match `task_id`

- AIR002
  A Dag should have
  an explicit `schedule` argument

AIR001

Example

```python
from airflow.operators import PythonOperator


incorrect_name = PythonOperator(task_id="my_task")
```

Use instead:

```python
from airflow.operators import PythonOperator


my_task = PythonOperator(task_id="my_task")
```

AIR002

Example

```python
from airflow import DAG


# Using the implicit default schedule.
dag = DAG(dag_id="my_dag")
```

Use instead:

```python
from datetime import timedelta

from airflow import DAG


dag = DAG(dag_id="my_dag", schedule=timedelta(days=1))
```

Ruff AIR Rules

Required changes for migrating to Airflow 3 ⛔

- AIR301
  Something removed in Airflow 3

- AIR302
  Something moved from Ariflow core
  to Airflow provider in Airflow 3

AIR301

Example

```
from airflow.utils.dates import days_ago


yesterday = days_ago(today, 1)
```

Use instead:

```
from datetime import timedelta


yesterday = today - timedelta(days=1)
```

AIR302

Example

```
from airflow.auth.managers.fab.fab_auth_manager import FabAuthManager

fab_auth_manager_app = FabAuthManager().get_fastapi_app()
```

Use instead:

```
from airflow.providers.fab.auth_manager.fab_auth_manager import FabAuthManager

fab_auth_manager_app = FabAuthManager().get_fastapi_app()
```

Ruff AIR Rules

Suggested changes for migrating to Airflow 3 ⚠️

- AIR311

  Similar to AIR301,

  but it won't break your code for the time being
  (We added a backward-compatible layer)

- AIR312

  Similar to AIR302,

  but it won't break your code for the time being
  (apache-airflow-providers-standard)

AIR311

Example

```
from airflow import Dataset


Dataset(uri="test://test/")
```

Use instead:

```
from airflow.sdk import Asset


Asset(uri="test://test/")
```

AIR312

Example

```python
from airflow.operators.python import PythonOperator


def print_context(ds=None, **kwargs):
    print(kwargs)
    print(ds)


print_the_context = PythonOperator(
    task_id="print_the_context", python_callable=print_context
)
```

Use instead:

```python
from airflow.providers.standard.operators.python import PythonOperator


def print_context(ds=None, **kwargs):
    print(kwargs)
    print(ds)


print_the_context = PythonOperator(
    task_id="print_the_context", python_callable=print_context
)
```

What AIR3XX can do today ✅

- Check renamed, moved, removed
  - variables
  - class attributes, methods
  - function arguments
  - airflow plugin extensions
- Check removed context key

What AIR3xx can't do today ❌

Open **Airflow 2 to 3 auto migration rules** #41641

Task • 9 / 10

- Code: AIR201
  - Add ruff rule to check the second argument of Asset/Dataset #48389
- Code: AIR301
  - 45961
    - `create_dagrun` removed from `airflow...DAG`
    - context key dag_run.external_trigger removed
  - 43774
    - Calling HookLineageCollector.create_asset with positional argument should raise an error
  - 45960
    - context_key inlet_events.uri → inlet_events.asset.uri
  - 41348
    - context key triggering_dataset_events → triggering_asset_events
  - 46415
    - Signature of airflow.models.baseoperatorlink.BaseOperatorLink.get_link changed

What AIR3xx can't do today ❌

- Access deprecated names using attribute access after importing the module
  (linting ✅, fixing ❌)

```
from airflow import decorators


decorators.dag
```

What AIR3xx can't do today ❌

- Access context variable in functions not decorated by @task

```
def print_context(**kwargs):
    context = get_current_context()
    print(context.get("conf"))


print_the_context = PythonOperator(
    task_id="print_the_context",
    python_callable=print_context,
)
```

What AIR can't do ❌

- Dynamically generated code

```
from airflow.sdk import DAG

example_dag = DAG(
    dag_id="example_dag",
    **{
        "fail" + "_stop": ...,
    },
)
```

Ch 3 - How is it done?

It all starts from ...

[LAZY CONSENSUS] Use Ruff as the official linter to detect Airflow 3 incompatibilities in user code

Posted to dev@airflow.apache.org

Tzu-ping Chung - Thursday, November 21, 2024 8:54:13 PM GMT+9

Hi all,

I am calling a LAZY CONSENSUS to officially anoint Ruff as the tool to implement lint rules that detect potential incompatibilities in existing user code (including DAG files and plugins) that may break in Airflow 3. The rationale is

* Ruff provides the best set of features for our needs according to my research, most importantly import resolution, and also a nice CLI “for free”.

* While a rule in Ruff must be implemented in Rust, which can be a hurdle to most Airflow contributors, it does not actually require a very deep understanding to Rust, but more toward the Python AST parser that Ruff provides (which is very similar to Python’s built-in ast).

* We intend to only implement lint rules, not “fixes” (which would allow Ruff to rewrite offending files automatically. This reduces the learning curve even further.

* Most Airflow contributors likely don’t know Python’s ast library very well to begin with anyway, so the learning curve would not be much flatter even if we choose a Python-based tool.

* We would not be the first Python library to use Ruff for this—Numpy also uses Ruff to detect potential incompatibilities for Numpy 2.0. We are also not the first to create Airflow rules in Ruff either—it already contains one Airflow-specific rule.

If we are to implement an auto fixer, it can still be rewritten in Python. With Ruff handling most of the user interaction (including generating proper error messages and a clear error code), the fixer can have a relatively simplistic CLI that can be more easily implemented with just argparse and print() (or a little drop of Rich if we feel fancy).

I have created a POC on generating lint errors against creating an Airflow DAG without using the 'schedule' argument correctly for Airflow 3: https://github.com/uranusjr/ruff/pull/1

If no one objects, the vote will pass three days from now, at 12:00 (noon) UTC, 24 November 2024. I intend to formally submit the rule implemented above to Ruff, and start contributing other rules to Ruff. A list of rule candidates can be found in Wei’s thread “[Discuss] Airflow 2 to 3 migration rules” sent to this list earlier.

Thanks all,
TP

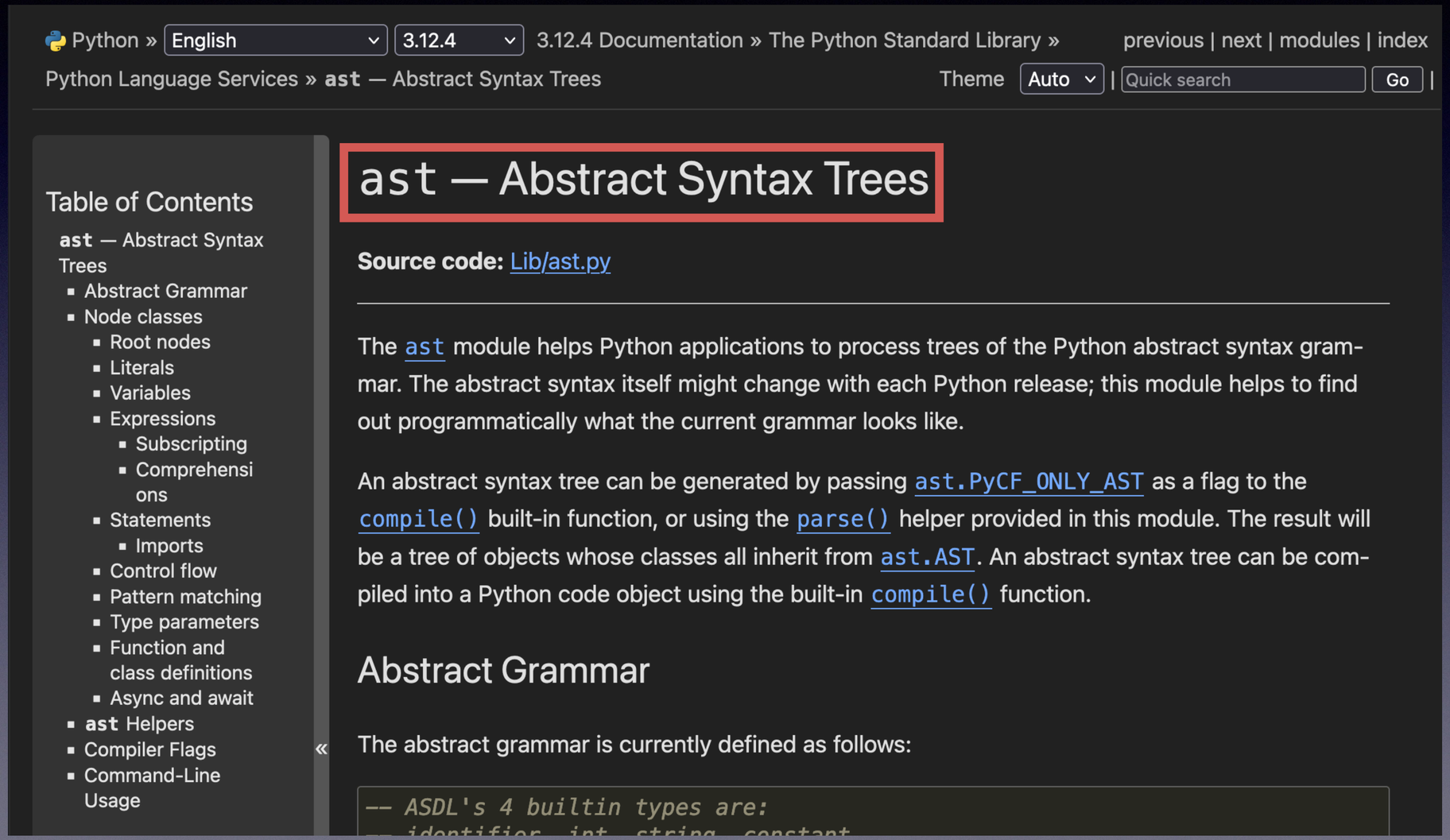
Why 🙋

- Ruff TUI is great!
- NumPy also uses Ruff to migrate to NumPy 2 (NPY201)
- There was an existing Airflow rule (AIR101)

How 🤔

`ast`

What is ast anyway?



Python » English 3.12.4 3.12.4 Documentation » The Python Standard Library » previous | next | modules | index

Python Language Services » ast — Abstract Syntax Trees Theme Auto | Quick search Go |

Table of Contents

- ast — Abstract Syntax Trees
  - Abstract Grammar
  - Node classes
    - Root nodes
    - Literals
    - Variables
    - Expressions
      - Subscripting
      - Comprehensions
    - Statements
      - Imports
    - Control flow
    - Pattern matching
    - Type parameters
    - Function and class definitions
    - Async and await
  - ast Helpers
  - Compiler Flags
  - Command-Line Usage

«

ast — Abstract Syntax Trees

Source code: Lib/ast.py

The ast module helps Python applications to process trees of the Python abstract syntax grammar. The abstract syntax itself might change with each Python release; this module helps to find out programmatically what the current grammar looks like.

An abstract syntax tree can be generated by passing ast.PyCF_ONLY_AST as a flag to the compile() built-in function, or using the parse() helper provided in this module. The result will be a tree of objects whose classes all inherit from ast.AST. An abstract syntax tree can be compiled into a Python code object using the built-in compile() function.

Abstract Grammar

The abstract grammar is currently defined as follows:

```
-- ASDL's 4 builtin types are:
-- identifier, int, string, constant
```

What is ast anyway?

A way to represent your code

in a tree structure

Minimum AST example

```python
import ast

source_code = "a = 1 + 2"

ast.dump(
    ast.parse(source_code),
    indent=4
)
```

Minimum AST example

```python
import ast

source_code = "a = 1 + 2"

ast.dump(
    ast.parse(source_code),
    indent=4
)
```

Minimum AST example

```python
import ast

source_code = "a = 1 + 2"

ast.dump(
    ast.parse(source_code),
    indent=4
)
```

Minimum AST example

```python
import ast

source_code = "a = 1 + 2"

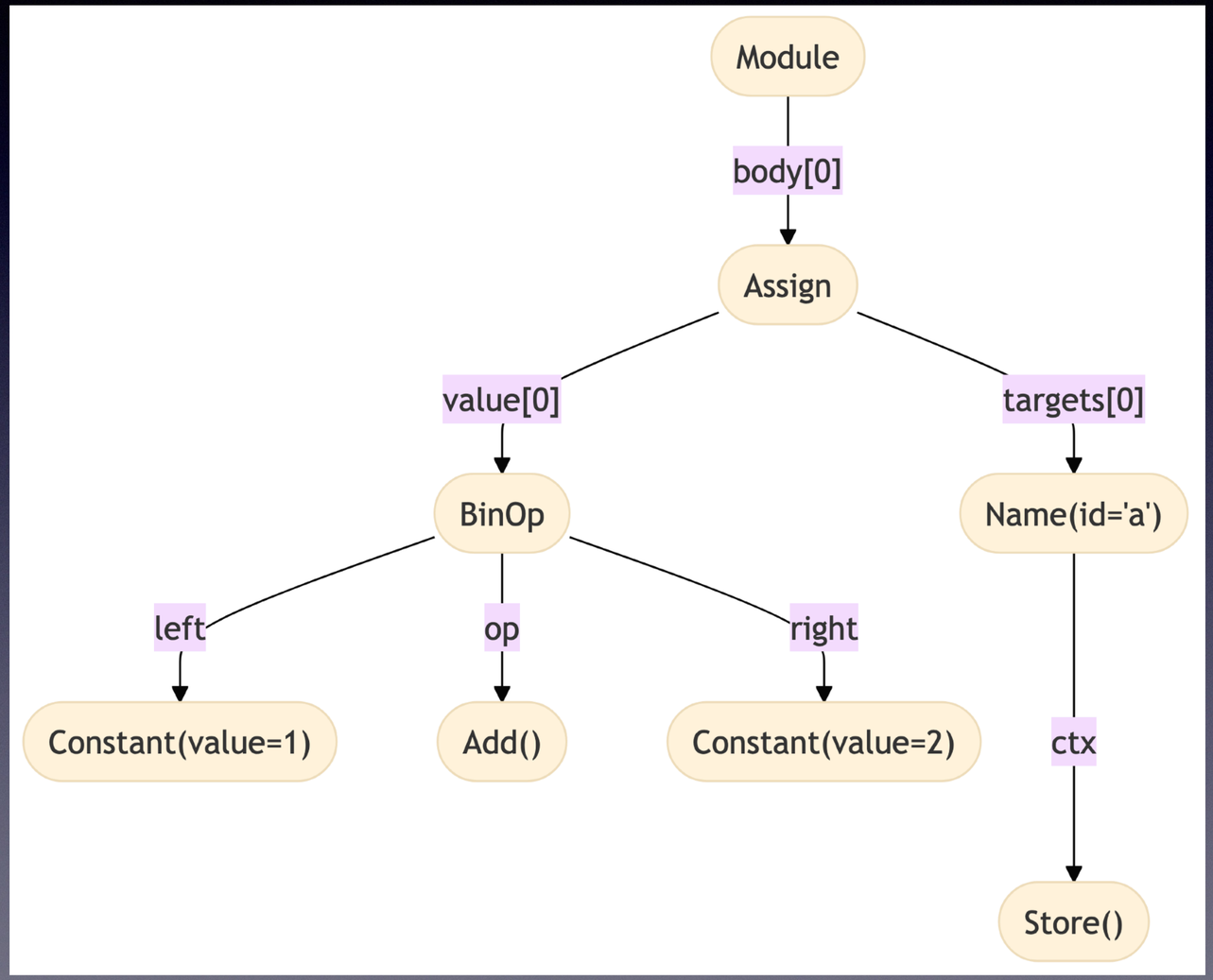
ast.dump(
    ast.parse(source_code),
    indent=4
)
```

Minimum AST example

```
Module(
    body=[
        Assign(
            targets=[
                Name(id="a", ctx=Store())
            ],
            value=BinOp(
                left=Constant(value=1),
                op=Add(),
                right=Constant(value=2)
            )
        )
    ],
    type_ignores=[]
)
```

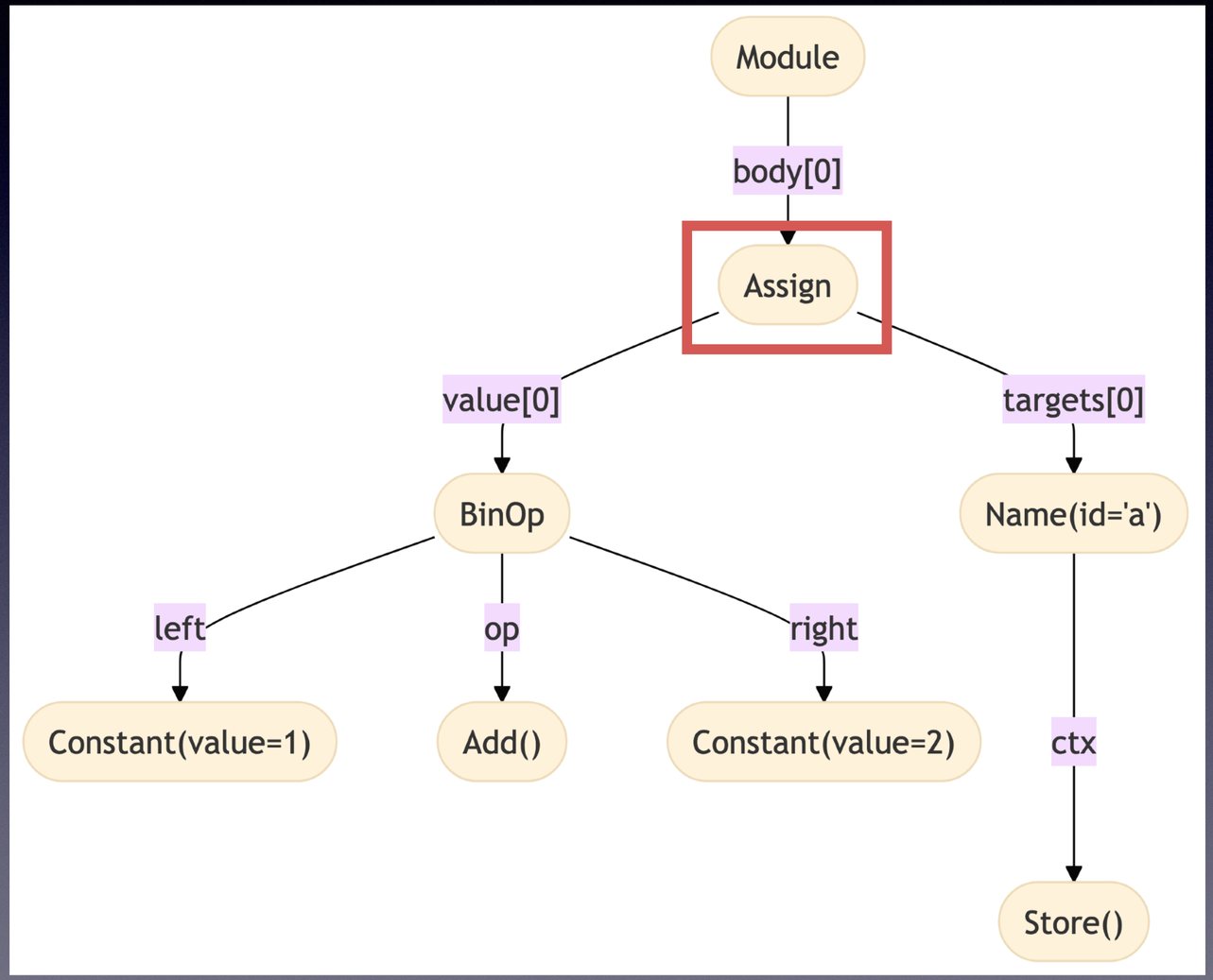
Minimum AST example

```
a = 1 + 2
```



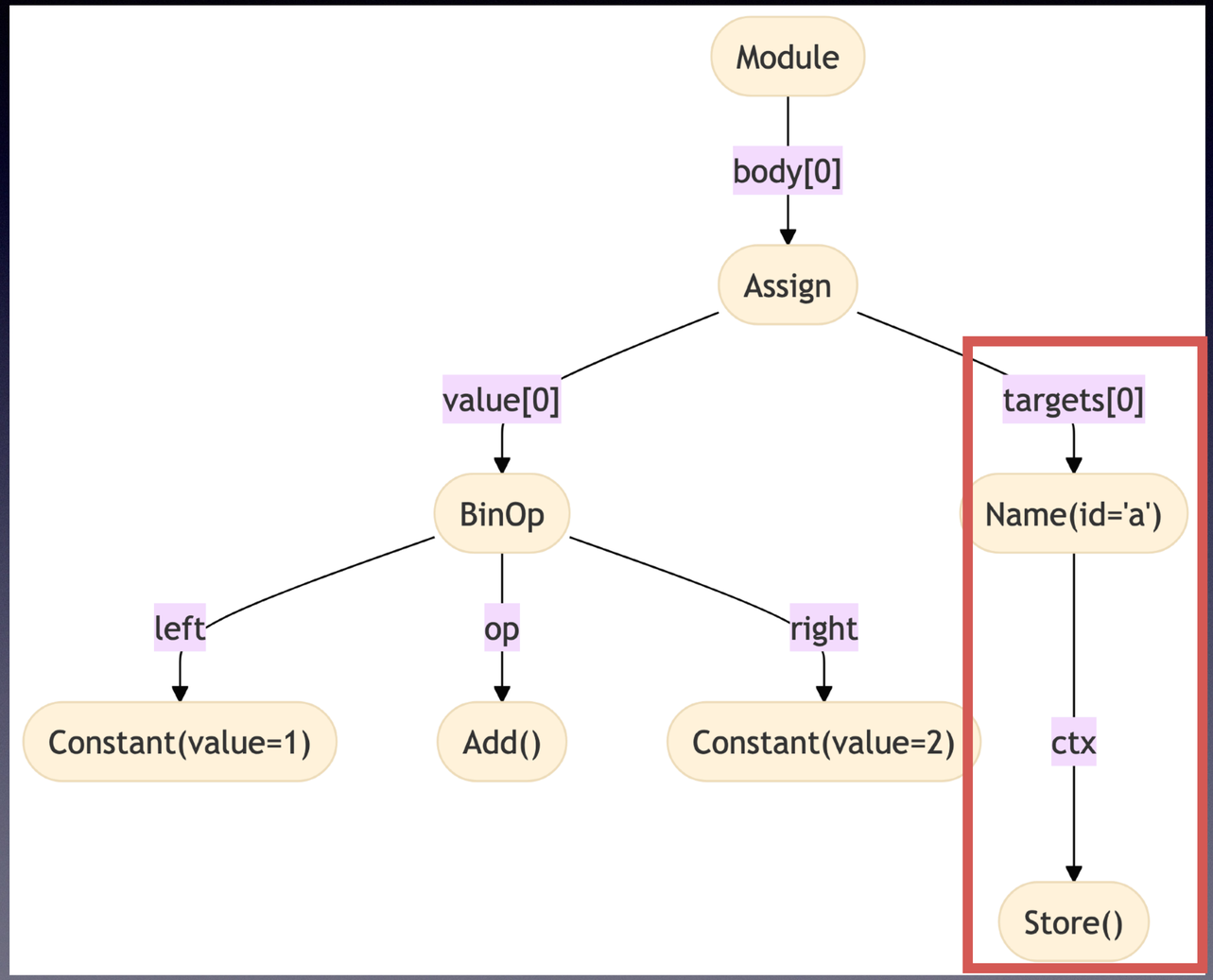
Minimum AST example

```
a = 1 + 2
```



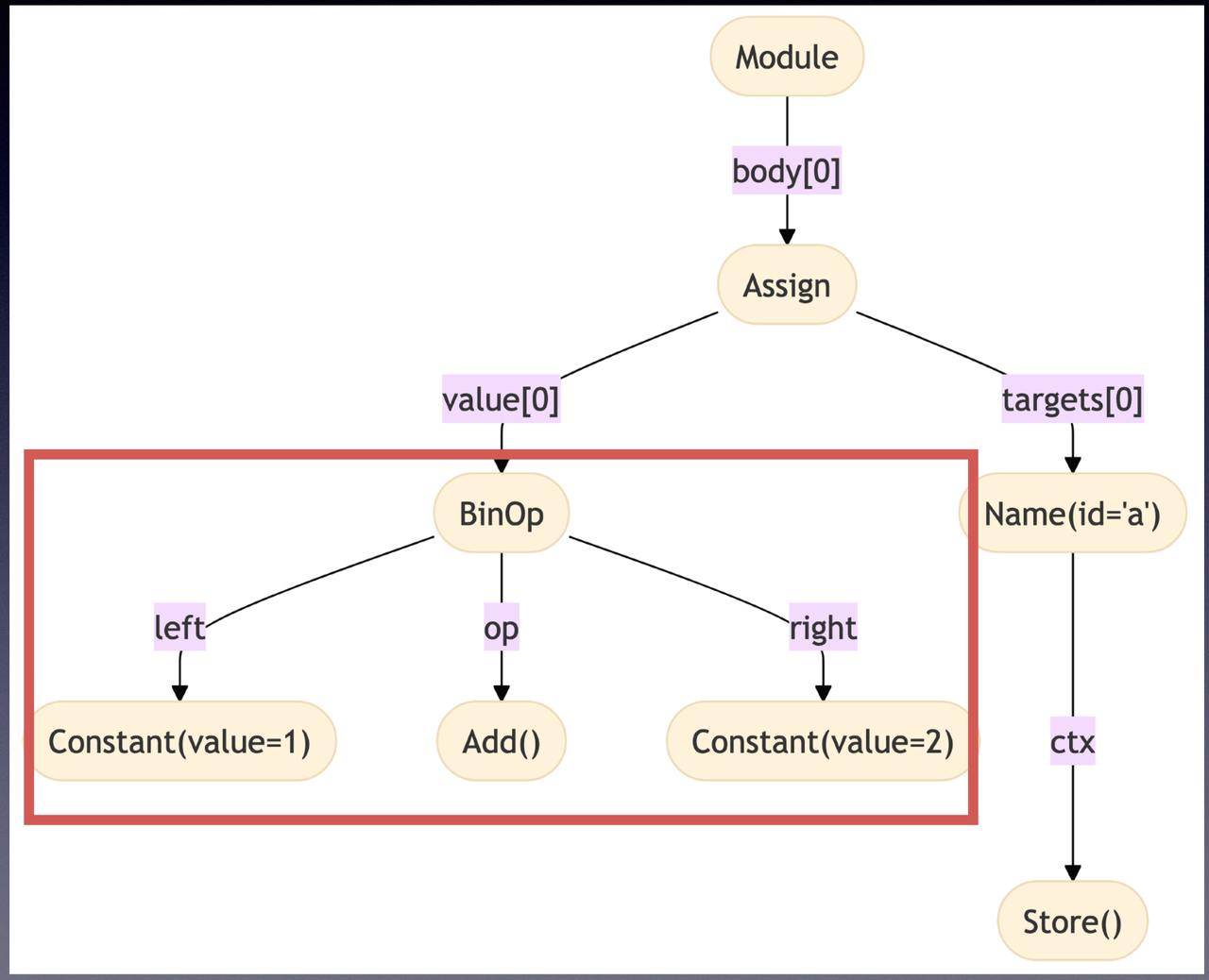
Minimum AST example

```
a = 1 + 2
```



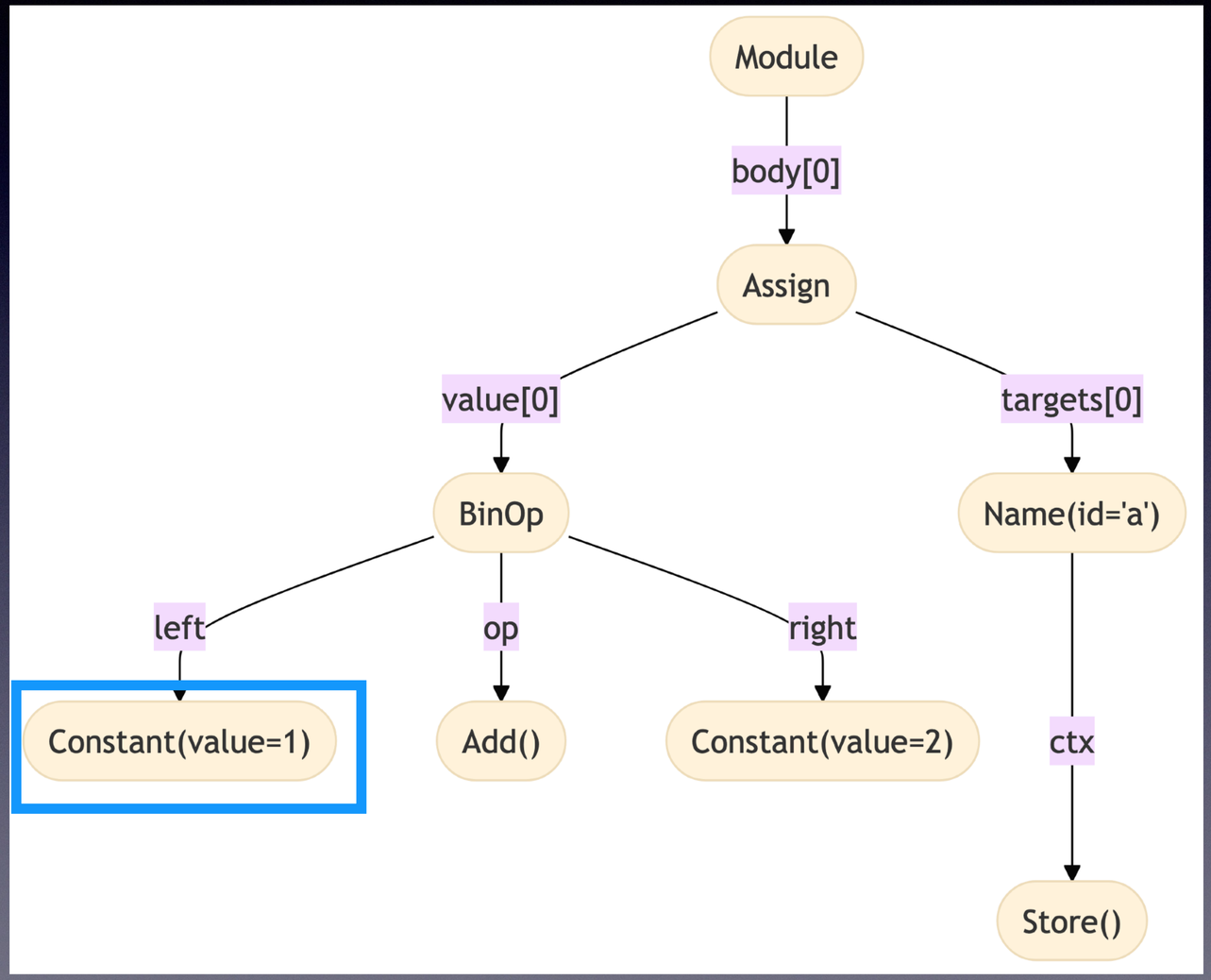
Minimum AST example

```
a = 1 + 2
```



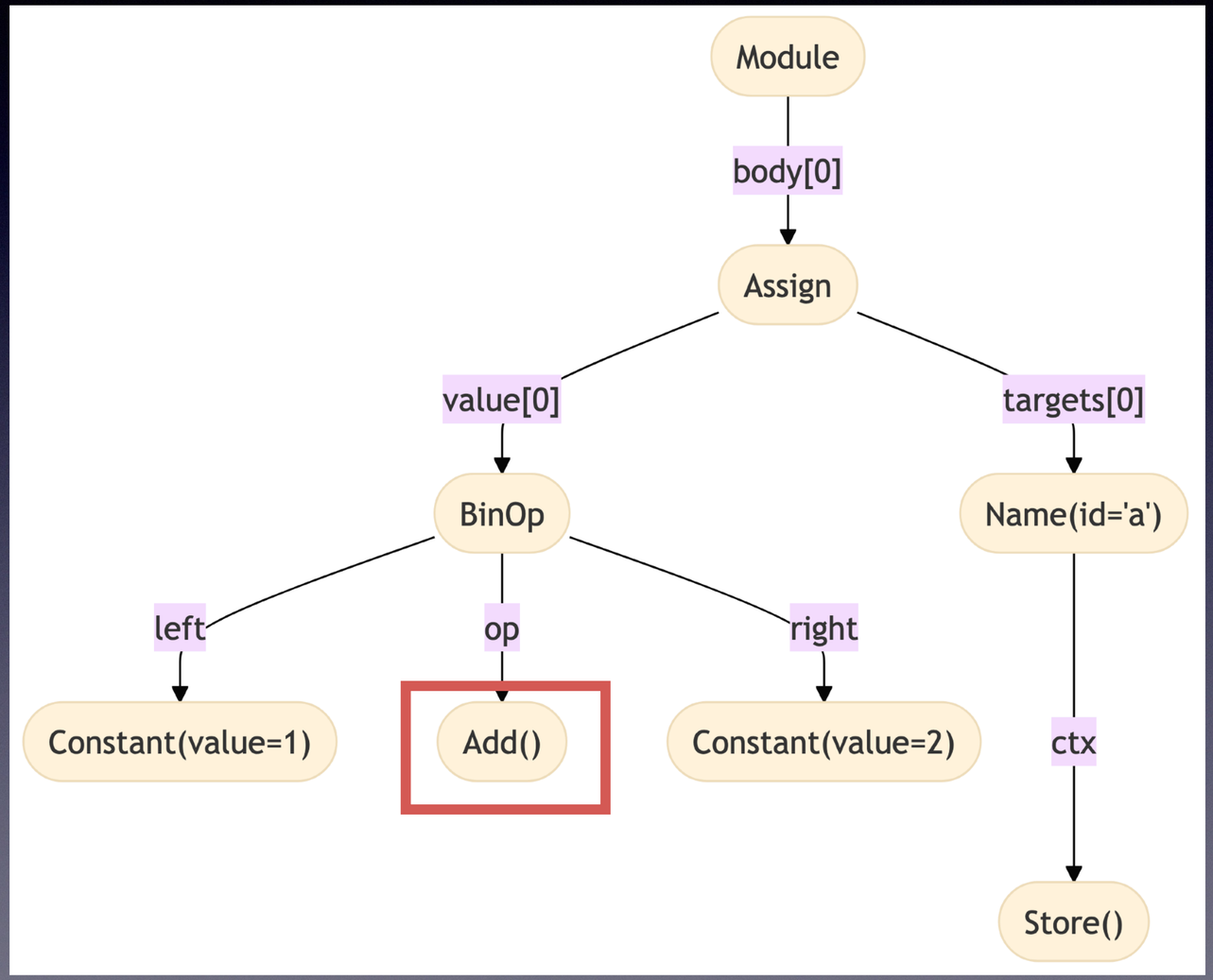
Minimum AST example

```
a = 1 + 2
```



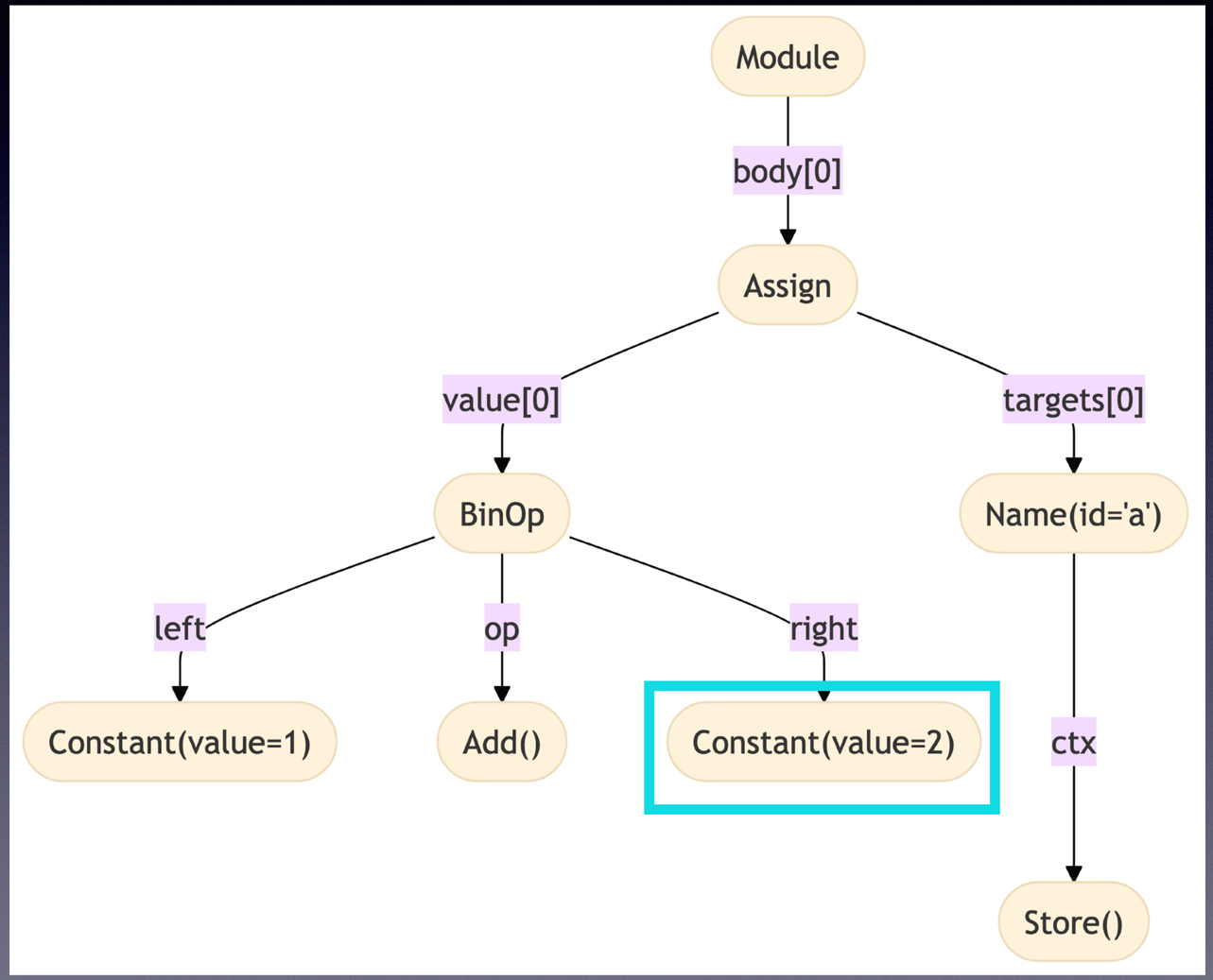
Minimum AST example

```
a = 1 + 2
```

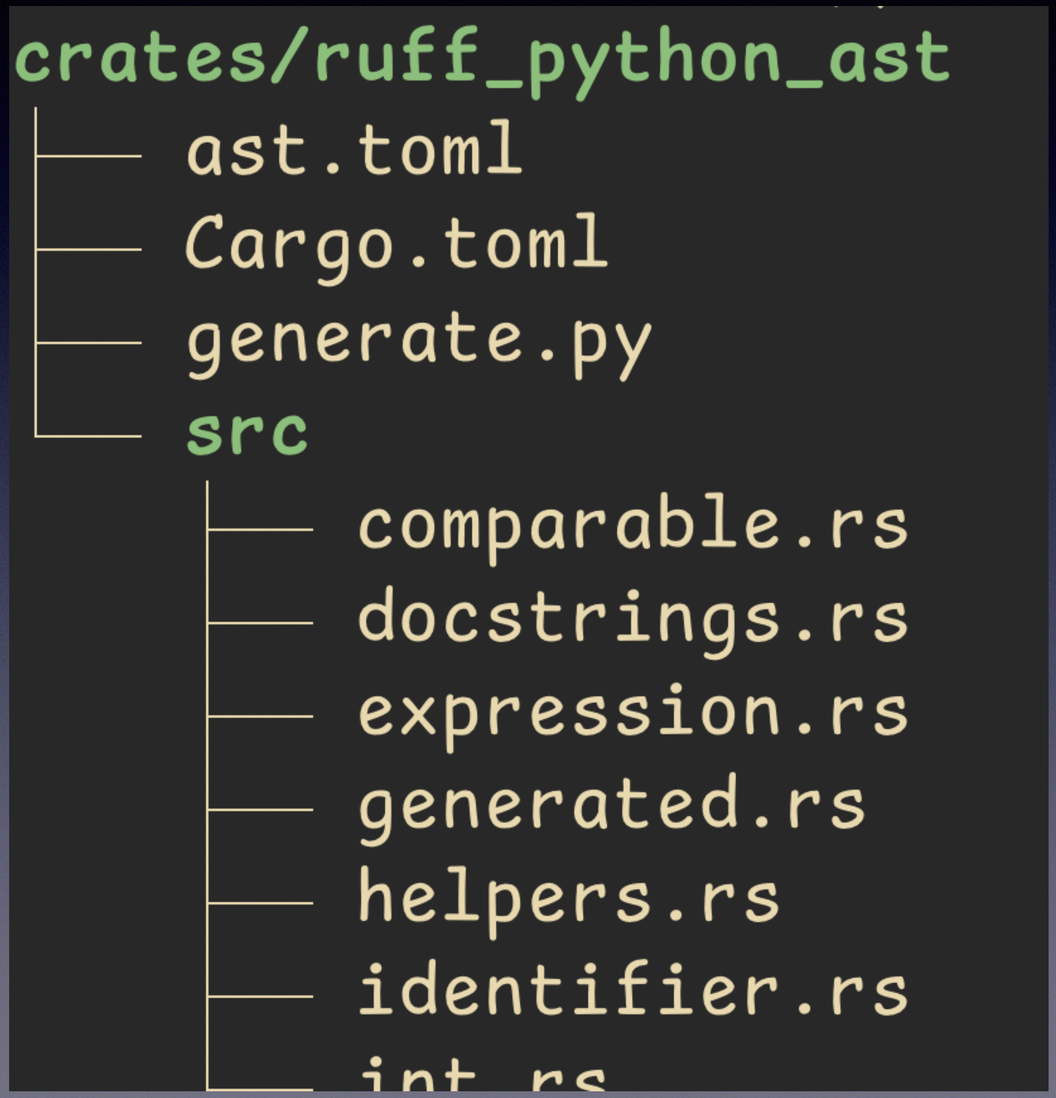


Minimum AST example

```
a = 1 + 2
```



Ruff also uses ast



🗄️ Files we care about when adding a Ruff rule

Adding new rules

- crates/ruff_linter/src/codes.rs

codes.rs

```
1028    pub fn code_to_rule(linter: Linter, code: &str) -> Option<(RuleGroup, Rule)>
1023        Some(match (linter, code) {
   8     (Flake8Todos, "007") => (RuleGroup::Stable, rules::flake8_todos::rules::Mis
   7     ↴
   6     // airflow↴
   5     (Airflow, "001") => (RuleGroup::Stable, rules::airflow::rules::AirflowVaria
   4     (Airflow, "002") => (RuleGroup::Stable, rules::airflow::rules::AirflowDagNo
   3     (Airflow, "301") => (RuleGroup::Stable, rules::airflow::rules::Airflow3Remo
   2     (Airflow, "302") => (RuleGroup::Stable, rules::airflow::rules::Airflow3Move
   1     (Airflow, "311") => (RuleGroup::Stable, rules::airflow::rules::Airflow3Sugg
1119     (Airflow, "312") => (RuleGroup::Stable, rules::airflow::rules::Airflow3Sugg
```

Register logic for ast nodes

```
crates/ruff_linter/src/checkers/ast/analyze/
├── bindings.rs
├── comprehension.rs
├── deferred_for_loops.rs
├── deferred_lambdas.rs
├── deferred_scopes.rs
├── definitions.rs
├── except_handler.rs
├── expression.rs
├── mod.rs
├── module.rs
├── parameter.rs
├── parameters.rs
├── statement.rs
├── string_like.rs
├── suite.rs
└── unresolved_references.rs
```

Checking/Fixing Logic

```
crates/ruff_linter/src/rules/airflow/
├── helpers.rs
├── mod.rs
├── rules
│   ├── dag_schedule_argument.rs
│   ├── mod.rs
│   ├── moved_to_provider_in_3.rs
│   ├── removal_in_3.rs
│   ├── suggested_to_move_to_provider_in_3.rs
│   ├── suggested_to_update_3_0.rs
│   └── task_variable_name.rs
└── snapshots
    ├── ruff_linter__rules__airflow__tests__AI
    └── ruff_linter__rules__airflow__tests__AI
```

Test cases

```
crates/ruff_linter/resources/test/fixtures/airflow/
├── AIR001.py
├── AIR002.py
├── AIR301_airflow_plugin.py
├── AIR301_args.py
├── AIR301_class_attribute.py
├── AIR301_context.py
├── AIR301_names_fix.py
```

e.g., AIR311 with --unsafe-fixes

```
from airflow.decorators import dag
```



```
from airflow.sdk import dag
```

How is this import detected and replaced?

```
from airflow.decorators import dag
```



```
from airflow.sdk import dag
```

1️⃣ Register the logic

expression.rs

```
30              Expr::Name(ast::ExprName {
24                  match ctx {
23                      ExprContext::Load => {
 4  if checker.is_rule_enabled(rule: Rule::Airflow3Removal) {
 3      airflow::rules::airflow_3_removal_expr(checker, expr);
 2  }
 1  if checker.is_rule_enabled(rule: Rule::Airflow3SuggestedUpdate) {
258      airflow::rules::airflow_3_0_suggested_update_expr(checker, expr);
 1  }
```

1️⃣ Register the check

expression.rs

```
Expr::Name(ast::ExprName {
    match ctx {
        ExprContext::Load => {
```

```
if checker.is_rule_enabled(rule: Rule::Airflow3Removal) {
    airflow::rules::airflow_3_removal_expr(checker, expr);
}
if checker.is_rule_enabled(rule: Rule::Airflow3SuggestedUpdate) {
    airflow::rules::airflow_3_0_suggested_update_expr(checker, expr);
}
```

1️⃣ Register the check

expression.rs

```
30              Expr::Name(ast::ExprName {
24                  match ctx {
23                      ExprContext::Load => {
 4      if checker.is_rule_enabled(rule: Rule::Airflow3Removal) {
 3          airflow::rules::airflow_3_removal_expr(checker, expr);
 2      }
 1      if checker.is_rule_enabled(rule: Rule::Airflow3SuggestedUpdate) {
258          airflow::rules::airflow_3_0_suggested_update_expr(checker, expr);
 1      }
```

1️⃣ Register the check

expression.rs

```
30              Expr::Name(ast::ExprName {
24                  match ctx {
23                      ExprContext::Load => {
 4  if checker.is_rule_enabled(rule: Rule::Airflow3Removal) {
 3      airflow::rules::airflow_3_removal_expr(checker, expr);
 2  }
 1  if checker.is_rule_enabled(rule: Rule::Airflow3SuggestedUpdate) {
258      airflow::rules: airflow_3_0_suggested_update_expr(checker, expr)
 1  }
```

2 Check based on expr type

suggested_to_update_3_0.rs

```rust
/// AIR311
pub(crate) fn airflow_3_0_suggested_update_expr(
    checker: &Checker,
    expr: &Expr
) {
    if !checker.semantic().seen_module(Modules::AIRFLOW) {
        return;
    }

    match expr {
        Expr::Call(ExprCall { ↙ 1
        }) => { ↙ 4
            Expr::Attribute(ExprAttribute { attr, .. }) => {
                check_name(checker, expr, attr.range());
            }
            Expr::Name(ExprName { ↙ 4
            }) => {
                check_name(checker, expr, *range);
            }
            _ => {}
    }
}
```

2 Check based on expr type

```rust
if !checker.semantic().seen_module(Modules::AIRFLOW) {
    return;
}
```

```rust
match expr {
    Expr::Call(ExprCall { ↙ 1
    }) => { ↙ 4
    Expr::Attribute(ExprAttribute { attr, .. }) => {
        check_name(checker, expr, attr.range());
    }
    Expr::Name(ExprName { ↙ 4
    }) => {
        check_name(checker, expr, *range);
    }
    _ => {}
}
```

2 Check based on expr type

```
match expr {
    Expr::Call(ExprCall { ↙ 1
    }) => { ↙ 4
    Expr::Attribute(ExprAttribute { attr, .. }) => {
        check_name(checker, expr, attr.range());
    }
    Expr::Name(ExprName { ↙ 4
    }) => {
        check_name(checker, expr, *range);
    }
    _ => {}
}
```

3 Resolve the fully qualified name

```
fn check_name( ↙ 1
) {
    let semantic = checker.semantic();

    let Some(qualified_name) = semantic
        .resolve_qualified_name(expr)
    else {
        return;
    };
```

Qualified name

PEP 3155 – Qualified name for classes and functions

Author:	Antoine Pitrou <solipsis at pitrou.net>
Status:	Final
Type:	Standards Track
Created:	29-Oct-2011
Python-Version:	3.3
Post-History:	
Resolution:	Python-Dev message

Qualified name

```
In [1]: from airflow.sdk import dag as not_dag

In [2]: not_dag.__qualname__
Out[2]: 'dag'
```

4️⃣ Match and generate a replacement object

```rust
let replacement = match qualified_name.segments() {
    // airflow.decorators
    [
        "airflow",
        "decorators",
        rest @ (
            "dag" | "task" | "task_group" | "setup" | "teardown"
        ),
    ] => Replacement::SourceModuleMoved {
        module: "airflow.sdk",
        name: (*rest).to_string(),
    },
```

4️⃣ Match and generate a replacement object

```rust
let replacement = match qualified_name.segments() {
    // airflow.decorators
    [
        "airflow",
        "decorators",
        rest @ (
            "dag" | "task" | "task_group" | "setup" | "teardown"
        ),
    ] => Replacement::SourceModuleMoved {
        module: "airflow.sdk",
        name: (*rest).to_string(),
    },
```

4️⃣ Match and generate a replacement object

```rust
let replacement = match qualified_name.segments() {
    // airflow.decorators
    [
        "airflow",
        "decorators",
        rest @ (
            "dag" | "task" | "task_group" | "setup" | "teardown"
        ),
    ] => Replacement::SourceModuleMoved {
        module: "airflow.sdk",
        name: (*rest).to_string(),
    },
```

4️⃣ Match and generate a replacement object

```rust
let replacement = match qualified_name.segments() {
    // airflow.decorators
    [
        "airflow",
        "decorators",
        rest @ (
            "dag" | "task" | "task_group" | "setup" | "teardown"
        ),
    ] => Replacement::SourceModuleMoved {
        module: "airflow.sdk",
        name: (*rest).to_string(),
    },
```

5️⃣ Early return if no fix needed

```
let (module, name) = match &replacement {
    Replacement::Rename { module, name } => (
        module, *name
    ),
    Replacement::SourceModuleMoved { module, name } => (
        module, name.as_str()
    ),
    _ => {
        checker.report_diagnostic(
            Airflow3SuggestedUpdate {
                deprecated: qualified_name.to_string(),
                replacement: replacement.clone(),
            },
            range,
        );
        return;
    }
};
```

6️⃣ Don't fix if it's in a try block

```
if is_guarded_by_try_except( ↙ 4
) {↴
    return;↴
}↴
```

6️⃣ Don't fix if it's in a try block

```python
try:
    from airflow.sdk import Asset
except ModuleNotFoundError:
    from airflow.datasets import Dataset as Asset
```

7️⃣ Create a diagnostic

```rust
let mut diagnostic = checker.report_diagnostic(
    Airflow3SuggestedUpdate {
        deprecated: qualified_name.to_string(),
        replacement: replacement.clone(),
    },
    range,
);
```

7️⃣ Create a diagnostic

```
AIR311 `airflow.decorators.dag` is removed in Airflow 3.0; It still works in
Airflow 3.0 but is expected to be removed in a future version.
 --> example.py:3:1
  |
1 | from airflow.decorators import dag
2 |
3 | dag
  | ^^^
  |
help: Use `dag` from `airflow.sdk` instead.

Found 1 error.
No fixes available (1 hidden fix can be enabled with the `--unsafe-fixes` opt
ion).
```

8️⃣ Set the fix

```
if let Some(fix) = generate_import_edit( ↙ 2
    ···.or_else(↴
        ···    || generate_remove_and_runtime_import_edit( ↙ 2
    ···)↴
{↴
    ···diagnostic.set_fix(fix);↴
}↴
```

8️⃣ Set the fix - safe fix

1. imports the new name
2. updates where the import is referenced

(e.g., `airflow.datasts.Dataset` to `airflow.sdk.Asset`)

```
/// Return the [`Fix`] that ↴
/// imports the new name and updates where the import is referenced.
/// This is used for cases that member name has changed.↴
/// (e.g., `airflow.datasts.Dataset` to `airflow.sdk.Asset`)↴
pub(crate) fn generate_import_edit(↴
```

8️⃣ Set the fix - unsafe fixes

1. removes the original import ⚠️
2. imports the same name with a new path

(e.g., `airflow.operators.pig_operator.PigOperator` to
`airflow.providers.apache.pig.operators.pig.PigOperator`)

```
/// Return the [`Fix`] that
/// remove the original import and import the same name with new path.
/// This is used for cases that member name has not changed.
/// (e.g., `airflow.operators.pig_operator.PigOperator` to
///  `airflow.providers.apache.pig.operators.pig.PigOperator`)
pub(crate) fn generate_remove_and_runtime_import_edit(
```

That's it 🏁

(the simplest example)

What's next ➡️


Explore and add static checks for DAGs for early detection of common issues #43176
Edit
New issue
Open
omkar-foss (Omkar P) opened on Oct 18, 2024
Collaborator
Description
As per users' feedback in the Airflow Debugging Survey 2024, 48.3% of respondents chose early issue detection during execution as one of their top 2 choices.
Use case/motivation
Start timer
Assignees
Lee-W
Labels
area:DAG-processing
kind:feature

$ cat weilee.py

```
__name__ = 李唯 / Wei Lee
__what_i_am_doing__ = [
    Software Engineer @ Astronomer,
    PMC Member @ Apache Airflow,
    Maintainer @ commitizen-tools,
    Volunteer @ PyCon Taiwan,
    Member @ Python Asia Organization,
    Meme Bot @ OpenSource4You,
    Mentor @ OpenSource4You,
]
__github__ = Lee-W
__linkedin__ = clleew
__site__ = https://wei-lee.me
```




Congratulations
New members of Project Management Committee
New committer

```
$ πthon weilee.py

  File "weilee.py", line 1
    __name__ = 李唯 / Wei Lee
                        ^^^

SyntaxError: invalid syntax
```

Python 3.14 was just released yesterday. 🎉

Tuesday, October 7, 2025

Python 3.14.0 (final) is here!



Python 3.14.0 is now available

Q & A

AIRFLOW SUMMIT

The 2025 Apache Airflow® Survey is here!

Fill it out to for a free Airflow 3 Fundamentals or DAG Authoring in Airflow 3 certification code

