

# Code quality in the repository

Anton Grishin ([@alchemmist](#))

EOSP #L4, winter 2026. CU x CPM



# Table of contents

---

`style` How to control code style in repo? Formatters and syle-checkers.

---

`lint` Set and follow some rules or best-practice in code base.

---

`types` How to check consistency of types system.

---

`CI` Automate all checks without installatino and configuring tools.

---

`make` Entry point into repository. Useful tool.

---

# What is style? style

```
import os,sys,math,random

def f(a,b=[]):
    x=0
    for i in a:
        if i==None:pass
        elif type(i)==int:x+=i
        else:x+=int(i)
    if b:
        for i in b:x+=i
    return x

from telegram.ext import TelegramBot

y=[1,"2",None,3]
z=[10,20]
print("Sum is "+str(f(y,z)))

# class d:
#     def __init__(self,n):self.n=n;self.l=[]
#     def add(self,v):self.l.append(v)
#     def get(self):
#         s=0
```

# Style it's about code formatting style

- The set of rules, that needs to be follow.
- For example: len of indent: tab, 4 space, 2 space, etc.
- Maximum line length (wrap long lines).
- Spaces around operators and after commas.
- Blank lines between functions/classes for readability.
- Order of imports (standard, third-party, local).
- The type of quotes (sigle or double).

cqfn/aibolit

style

The screenshot shows a GitHub issue page for the repository 'cqfn / aibolit'. The issue title is 'Inconsistent quote style in the codebase disrupting automated formatting #819'. The issue is marked as 'Closed' and has a link to the pull request #823. The issue was opened by 'alchemist' on June 12, 2025, and was last edited by 'alchemist'. The issue description discusses the inconsistency in quote styles (single vs. double) in the codebase and mentions the use of 'ruff' for automated formatting. A code snippet for the 'pyproject.toml' file is provided, showing the configuration for 'ruff' to use double quotes. The issue is categorized with labels 'bug', 'good-title', and 'help wanted'. The right sidebar shows that no one is assigned, no type is set, and no projects or milestones are associated with the issue.

cqfn / aibolit

Type to search

<> Code Issues 52 Pull requests 18 Actions Security Insights

## Inconsistent quote style in the codebase disrupting automated formatting #819

Edit New issue

Closed #823

alchemist opened on Jun 12, 2025 · edited by alchemist

Now I see in code base some chaos in quote. Some times (1, 2, 3) using single quote, some times (1, 2, 3) double. Of course, in Python it interchangeable, but holistic style requires one way for anything.

Now we already use `ruff` and his configuration have a feature for this:

```
# pyproject.toml

[tool.ruff.format]
quote-style = "double" # or single
```

My usecase: I'm setup my text editor for auto code formatting. It's useful! But now, if i do it, in my pull request will be many incomplete changes, that will complicate the review. Therefore, I want sync our code style.

Assignees: No one assigned

Labels: bug, good-title, help wanted

Type: No type

Projects: No projects

Milestone: No milestone

Relationships

# Painstaking work or magic? style

The screenshot shows a GitHub pull request page for the repository 'cqfn / aibolit'. The pull request title is 'Add ruff checks to enforce single quotes #823'. It was merged on Jun 14, 2025, by rultor, merging 3 commits from the branch 'AntonProkopyev:819-enforce-single-quotes' into 'cqfn:master'. The pull request has 15 conversations, 3 commits, 36 checks, and 110 files changed. A summary by CodeRabbit is visible, detailing changes in style, documentation, and chores. The style section includes standardizing string literals and adjusting formatting settings. The documentation section includes updating docstring styles. The chores section includes updating configuration files. The pull request also has reviewers: coderabbitai[bot], yegor256, and ivanovmg. A milestone is not set. The development section notes that successfully merging this pull request may close issues, including 'Inconsistent quote style in the codebase dis...'. Notifications can be customized.

cqfn / aibolit

Search Type to search

<> Code Issues 52 Pull requests 18 Actions Security Insights

## Add ruff checks to enforce single quotes #823

<> Code

Merged rultor merged 3 commits into cqfn:master from AntonProkopyev:819-enforce-single-quotes on Jun 14, 2025

Conversation 15 Commits 3 Checks 36 Files changed 110 +1,254 -1,245

AntonProkopyev commented on Jun 13, 2025 • edited by coderabbitai [bot] Contributor

Closes #819

### Summary by CodeRabbit

- **Style**
  - Standardized all string literals to use single quotes and updated docstring delimiters for consistency throughout the codebase.
  - Adjusted formatting settings to enforce these styles automatically.
- **Documentation**
  - Updated docstring styles across classes and methods for improved readability and consistency.
- **Chores**
  - Updated configuration files to enforce and automate new code style rules.

**Reviewers**

- coderabbitai[bot]
- yegor256
- ivanovmg

**Milestone**

No milestone

**Development**

Successfully merging this pull request may close these issues.

- ✓ Inconsistent quote style in the codebase dis...

**Notifications** Customize

# Code formatters for Python

- `black` — classic way (nondeterministic changes on edge cases).
- `blue` — more configurable, a little bit rules changed (written on Python — slow on big projects).
- `isort` — sorts imports into standard/third-party/local groups
- `ruff` — linter + formatter, fast and all-in-one (written on Rust ⚡)
- `autopep8` — fixes PEP8 violations automatically
- `yapf` — flexible formatting based on style config

The screenshot shows the Ruff project website. The navigation menu on the left includes: Ruff, Overview, Testimonials, Tutorial, Installing Ruff, The Ruff Linter, The Ruff Formatter, Editors, Configuring Ruff, Preview, Rules, Settings, Versioning, Integrations, FAQ, and Contributing. The main content area features a header with the Ruff logo, a search bar, and social media links for GitHub (0.15.2), GitHub stars (46k), and GitHub forks (1.8k). Below the header is a horizontal bar with various badges: Ruff, pypi v0.15.2, license MIT, python 3.7 | 3.8 | 3.9 | 3.10 | 3.11 | 3.12 | 3.13 | 3.14, CI passing, and Discord. The main heading is "Ruff", followed by "Docs | Playground". A descriptive sentence states: "An extremely fast Python linter and code formatter, written in Rust." Below this is a horizontal bar chart titled "Linting the CPython codebase from scratch." comparing Ruff (0.29s) to Autoflake (6.18s), Flake8 (12.26s), Pyflakes (15.79s), Pycodestyle (46.92s), and Pylint (> 60s). The chart shows Ruff is significantly faster than the other tools. Below the chart is a list of features:

- ⚡ 10-100x faster than existing linters (like Flake8) and formatters (like Black)
- 🐍 Installable via `pip`
- 🔧 `pyproject.toml` support
- 🍷 Python 3.14 compatibility
- 🔗 Drop-in parity with `Flake8`, `isort`, and `Black`

## Why Ruff? `style`

- Very fast on real repositories, so it is easy to run constantly.
- Deterministic formatting with minimal configuration overhead.
- One ecosystem for both formatting and linting in the same toolchain.
- Simple `uv` workflow: install once, run everywhere (local, hooks, CI).
- Actively maintained and widely adopted in modern Python projects.

# Ruff in action

 main.py

```
1 import os, sys, math, random
2
3
4 def calc_sum(a, b=[]):
5     total = 0
6     for i in a:
7         if i == None:
8             pass
9         else:
10            total += int(i)
11    if b:
12        total += sum(b)
13    return total
14
15
16 x = [1, "2", None, 3]
17 y = [10, 20]
18 print("Sum is " + str(calc_sum(x, y)))
19 print(math.pi, random.randint(1, 100))
```

We can check format of this file with:

```
$ uv run ruff format --check main.py
```

```
Would reformat: main.py
1 file would be reformatted
```

And format it automatically:

```
$ uv run ruff format --check main.py
```

```
1 file reformatted
```

# Running on project style

```
$ uv run ruff format . --check
```

```
Would reformat: src/lib_demo/report.py
```

```
12 files would be reformatted, 38 files already formatted
```

```
$ uv run ruff format .
```

```
12 files reformatted, 37 files left unchanged
```

# Configure Ruff formatter style

```
pyproject.toml
```

```
[tool.ruff]
line-length = 88
target-version = "py312"

[tool.ruff.format]
quote-style = "double"
indent-style = "space"
line-ending = "auto"
```

Install `uv` as devdependency:

```
uv add --dev ruff
```

Format entire project:

```
uv run ruff format .
```

Check without write changes:

```
uv run ruff format . --check
```

## What is linter? How it differs from formatter? `lint`

- Linter analyses code and reports violations of rules and best practices.
- Formatter rewrites code style automatically (spaces, quotes, wraps, etc.).
- Linter focuses on quality risks: unused imports, bugs, complexity, anti-patterns.
- Formatter answer: “How code looks?” Linter answer: “Is code safe and clean?”
- Some linter rules are auto-fixable, but many need developer decision.

## Popular Python linters `lint`

- `flake8` — classic plugin-based linter, many teams still use it.
- `pylint` — very strict analyzer, checks style and design smells deeply.
- `ruff` — very fast linter with large ruleset, drop-in for many flake8 plugins.
- `bandit` — security-oriented static checks for common Python vulnerabilities.

# Why Ruff is special <sup>lint</sup>

- `ruff` combines both roles: formatter (`ruff format`) and linter (`ruff check`).
- Single tool, single config, single install path via `uv`.
- Ruff implements rules inspired by multiple tools and plugins.
- Examples: Pyflakes (`F`), pycodestyle (`E/W`), isort (`I`), bugbear (`B`), pyupgrade (`UP`).
- Result: fewer tools in repo, same or stronger quality gates.

# Ruff in action again! lint

main.py

```
1 import math
2 import random
3
4
5 def calc_sum(a, b=[]):
6     total = 0
7     for i in a:
8         if i == None:
9             pass
10        else:
11            total += int(i)
12    if b:
13        total += sum(b)
14    return total
15
16
17 x = [1, "2", None, 3]
18 y = [10, 20]
19 print("Sum is " + str(calc_sum(x, y)))
20 print(math.pi, random.randint(1, 100))
```

Run checks on your code:

```
$ uv run ruff check main.py
```

```
main.py:7:17: E711 Comparison to `None` should be `cond is None`
|
5 |     total = 0
6 |     for i in a:
7 |         if i == None:
|             ~~~~ E711
8 |             pass
9 |         elif type(i) == int:
|
= help: Replace with `cond is None`
```

Try to fix someone:

```
$ uv run ruff check main.py --fix
```

```
Found 4 errors (3 fixed, 1 remaining).
No fixes available (1 hidden fix can be enabled with the `--unsafe-fixes` flag)
```

# Configure Ruff linter lint

```
pyproject.toml
```

```
[tool.ruff.lint]
select = ["ALL"]
ignore = ["D203", "D212", "COM812", "PLR0913"]

[tool.ruff.lint.per-file-ignores]
"tests/**/*.py" = ["S101", "D", "INP001", "ARG001", "ARG002"]
```

Install and run:

```
uv add --dev ruff
uv run ruff check .
uv run ruff check . --fix
```

With formatter in one flow:

```
uv run ruff check . --fix --unsafe-fixes
uv run ruff format .
```

# Let's see new pull request

live-demo

The screenshot shows a GitHub pull request page for the repository 'contriboo / contriboo-lib'. The title of the pull request is 'Make ruff stricter. Setup rules #39'. It was opened by 'alchemmist' and is currently open. The pull request description includes a section titled 'PR-Codex overview' and a 'Detailed summary' with a bulleted list of changes. The right sidebar shows various settings for the pull request, such as Reviewers, Assignees, Labels, Projects, Milestone, and Development.

contriboo / contriboo-lib

Type to search

<> Code Issues 26 Pull requests 2 Discussions Actions Projects Wiki Security Insights Settings

## Make ruff stricter. Setup rules #39

Open alchemmist wants to merge 12 commits into main from 38

Conversation 9 Commits 12 Checks 6 Files changed 27 +705 -192

alchemmist commented 9 hours ago • edited by pr-codex bot

Close #38

### PR-Codex overview

This PR focuses on enhancing the contriboo project by updating linting, type checking, and security checks in CI workflows, improving type hinting and error handling, and refactoring code for better readability and maintainability.

### Detailed summary

- Updated CI workflows to use ruff, bandit, and mypy directly.
- Refactored type aliases to type definitions in src/contriboo/profile/types.py.
- Improved error handling in exceptions.py with specific error classes.
- Enhanced logging in ProfileAnalysisService.
- Refactored several methods for clarity and consistency.
- Added type hints and improved type checking in various files.

Reviewers: coderabbitai[bot]

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Development

# Static analyzers (recap) `types`

We already covered theory in previous lecture, here is a practical map.

- `mypy` : reference checker, huge ecosystem and many plugins/extensions.
- `pyright` / `basedpyright` : very fast, strict defaults, strong IDE feedback.
- `pyre` : Meta checker, good performance, mostly used in specific stacks.
- `pytype` : Google checker with deep inference, less common in OSS tooling today.
- `ty` (Astral team): promising new checker from the same team, but still early and raw.
- In practice: choose one primary checker and run it everywhere via `uv` + CI.

## Why mypy in this course? `types`

- Mature and stable: predictable behavior for students and CI.
- Rich ecosystem of plugins/extensions for real frameworks and libraries.
- Lots of docs, examples, and community answers for every typical issue.
- Strict mode gives clear quality gate for typed Python in educational projects.
- We can add `pyright` as second opinion, but keep `mypy` as baseline.
- `ty` is interesting, but currently too early to be course baseline.

# Type checker configs (short) types

pyproject.toml (mypy):

```
[tool.mypy]
python_version = "3.12"
strict = true
warn_unused_ignores = true
warn_return_any = true
disallow_untyped_defs = true
```

pyrightconfig.json:

```
{
  "pythonVersion": "3.12",
  "typeCheckingMode": "strict",
  "include": ["src"],
  "exclude": ["tests/fixtures"]
}
```

## Run static analysis with `uv` run

```
$ uv add --dev mypy basedpyright
$ uv run mypy src
Success: no issues found in 27 source files

$ uv run basedpyright
0 errors, 0 warnings, 0 notes
```

# What is `make`?

- `make` is a task runner built around named targets in `Makefile`.
- Target is a command alias: `make lint`, `make test`, `make check`.
- Targets can depend on each other: one command can run a full pipeline.
- It standardizes local workflow, onboarding, and CI commands.

# contriboo-lib Makefile structure make

Core quality flow:

```
check: lint types format-check test

lint:
  uv run ruff check src tests && markdownlint-cli2 .

types:
  uv run mypy --strict ... src tests

format:
  uv run ruff format && uv run ruff check --fix -s

test:
  uv run pytest -q
```

Other groups:

- Security: `security`
- Tests: `test-cov`
- Setup: `venv`, `requirements`
- Hooks: `pre-commit`, `pre-commit-install`, `pre-commit-uninstall`
- Utility: `clean`, `help`

Important internals:

```
.DEFAULT_GOAL := help
SHELL := /bin/bash
.SHELLFLAGS := -eu -o pipefail -c
UV := uv
RUFF := $(UV) run ruff
```

# Make targets map (short) `make`

Use case	Targets
Main quality gate	<code>check</code>
Style and lint	<code>format</code> , <code>format-check</code> , <code>lint</code>
Typing	<code>types</code>
Tests	<code>test</code> , <code>test-cov</code>
Security	<code>security</code>
Setup	<code>venv</code> , <code>requirements</code>
Hooks	<code>pre-commit</code> , <code>pre-commit-install</code> , <code>pre-commit-uninstall</code>
Utility	<code>clean</code> , <code>help</code>

# Example `make` runs from `contriboo-lib` run

```
$ make check
Running linters...
uv run ruff check src tests
markdownlint-cli2 .
Lint completed!

Checking types...
uv run mypy --strict --disallow-untyped-defs --disallow-incomplete-defs src tests
Success: no issues found in 19 source files
Type check completed!

Checking formatting...
uv run ruff format --check
50 files already formatted
Format check completed!

Running tests...
uv run pytest -q
16 passed in 0.52s
Tests completed!
```

```
$ make security
uv run bandit -r src/
No issues identified.
```

# CI strategy in `contriboo-lib` ci

Workflows split by responsibility:

```
.github/workflows/ruff.yaml  
.github/workflows/mypy.yaml  
.github/workflows/pytest.yaml  
.github/workflows/coverage.yaml  
.github/workflows/security.yaml  
.github/workflows/markdownlint.yaml
```

Each one runs on `pull_request` .

Why split instead of one huge job:

- Fast feedback per signal (lint/types/tests/security).
- Easy to rerun failed check only.
- Simple ownership of pipeline parts.
- Clear mapping from badge/status to action.

# Real GitHub Actions examples <sup>ci</sup>

ruff.yaml :

```
name: ruff
on:
  pull_request:
jobs:
  quality:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v6
      - uses: astral-sh/setup-uv@v7
      - run: uv sync
      - run: uv run ruff check src tests
```

mypy.yaml :

```
name: mypy
on:
  pull_request:
jobs:
  types:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v6
      - uses: astral-sh/setup-uv@v7
      - run: uv sync
      - run: uv run mypy --strict --disallow-untyped-defs --disa
```

# Coverage + security workflows ci

```
name: pytest-coverage
on:
  pull_request:
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: astral-sh/setup-uv@v7
      - run: uv sync
      - run: |
          export PYTHONPATH=$PYTHONPATH:$(pwd)/src
          uv run pytest --cov=src --cov-report=term-missing:skip-covered --cov-fail-under=70 tests/
```

```
name: security
on:
  pull_request:
jobs:
  security:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v6
      - uses: astral-sh/setup-uv@v7
      - run: uv sync
```

# Local `make` and CI mapping <sup>ci</sup>

Local command	GitHub Actions workflow
---------------	-------------------------

<code>make lint</code>	<code>ruff.yaml</code> + <code>markdownlint.yaml</code>
------------------------	---

<code>make types</code>	<code>mypy.yaml</code>
-------------------------	------------------------

<code>make test</code>	<code>pytest.yaml</code>
------------------------	--------------------------

<code>make test-cov</code>	<code>coverage.yaml</code>
----------------------------	----------------------------

<code>make security</code>	<code>security.yaml</code>
----------------------------	----------------------------

<code>make check</code>	aggregate local gate before push
-------------------------	----------------------------------

# Summary

- `ruff format` gives deterministic style with minimal setup.
- Linters catch quality issues early: Ruff for Python, markdownlint for docs.
- Static analyzers are short in theory, strict in practice: `uv run mypy --strict ...``.
- `contriboo-lib`` Makefile covers full dev lifecycle: setup, checks, security, hooks, cleanup.
- CI in GitHub Actions mirrors local commands with focused workflows.

