Engineer

Ruff

🖌 a fast linter and formatter for Python

What it is

Ruff is a single command-line program, written in Rust, that checks Python source code against a catalogue of lint rules and rewrites its layout into one consistent style. It is backed by Astral, the company behind uv and ty, and two of its subcommands do the daily work: `ruff check` is the linter and `ruff format` is the formatter. It claims to run 10 to 100 times faster than Flake8 and Black, keeps a cache so that unchanged files are not analysed again, and can stand in for Flake8 with dozens of its plugins, Black, isort, pydocstyle, pyupgrade and autoflake.

The linter ships more than 900 rules, many of them native rewrites of popular Flake8 plugins. Every rule has a code made of a short prefix and three digits, and the prefix says where the rule came from: F401, an unused import, comes from Pyflakes, and E501, a line that runs too long, from pycodestyle. A configuration selects rules by those codes, one at a time or by prefix, so `select = ["E", "F"]` turns on every rule whose code starts with E or F, and the special selector `ALL` turns on everything. Many rules carry a fix, sorted into safe ones, meant to keep runtime behaviour, and unsafe ones, which might change it; `ruff check --fix` applies only the safe kind unless `--unsafe-fixes` is passed too.

The formatter is designed to take Black's place as a drop-in replacement. On codebases such as Django and Zulip more than 99.9% of lines come out identical, though the docs say plainly that the two are not meant to be swapped back and forth on the same code. Like Black, it skips regions fenced with `# fmt: off` and `# fmt: on`.

Both halves read the same configuration, which lives under `[tool.ruff]` in `pyproject.toml`, or in a `ruff.toml` or `.ruff.toml` file. Each Python file is checked against the closest configuration file above it, and that file is used whole: Ruff

merges nothing from parent directories unless the nearer file names its parent with `extend`. Options given on the command line override every file. Left unset, lines may run to 88 characters, as in Black, and the Python version the rules assume is taken from the project's `requires-python` field when no configuration names one.

Use the concept in a decision

Put "What is Ruff?" to work in a real decision. We can help relate the concept to the system you are building or buying. One team taking an idea from a sketch to something your customers can log into, instead of handing it between three suppliers.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Led a GIS map application to 10x revenue growth — 10x revenue growth, led from the map product

Led a GIS map application to 10x revenue growth Get in touch →

Why it matters

Speed decides where a check is allowed to run. A linter slow enough to interrupt someone gets pushed out to continuous integration, where it reports after the author has moved on. One that answers quickly can run in the editor and in a commit hook, while the change is still fresh.

Consolidation removes arguments between tools. A project that lints with Flake8 and plugins, sorts imports with isort and formats with Black has to keep several configurations and version pins in agreement. Ruff offers one configuration, one version to pin and one numbering of rules, and it publishes the list of lint rules that fight its own formatter, so the two can be made to agree on purpose.

Its exit codes make it a gate without a wrapper. `ruff check` exits with 0 when there are no violations left or every one was fixed, with 1 when violations remain, and with 2 when the configuration or the command line is invalid or Ruff itself fails, as ESLint and Prettier do. `ruff format --check` exits with 1 when any file would be reformatted. A rule is either selected or it is off, and any selected rule that fires sets the exit code to 1.

The cost is that the rule set is a decision someone has to own. Ruff's own advice is to list rules explicitly with `select` rather than stacking `extend-select`, to start from E and F and add one group at a time, and to use `ALL` with discretion, because it enables new rules on every upgrade.

What it looks like in practice

Ruff covered Python in a zero-warnings bar set across six languages, where each language had a linter of its own. Every diagnostic counted as an error and inline suppressions were banned, which for Ruff rules out the `# noqa` comment: a Python finding could be fixed in the code and in no other way. Pre-commit and pre-push hooks ran the linters and the tests, so the check ran before a commit existed.

At the documented October 2026 tooling stage, this site gave Ruff a smaller job: the 22 helper scripts beside the Hugo build. The credits page lists Ruff among the tools that keep the source honest. The rule set in `pyproject.toml` is measured rather than chosen. On 2026-10-15 every one of Ruff's 54 rule families was run against those scripts, 39 of them reported nothing, and the silent families whose subject is this code were added to the original eleven, which brings the selection to 33. The security family, S, stays on with five rules ignored by code, each beside a comment explaining why the flagged pattern here is the safe twin of a dangerous one, so the other rules in that family still gate. Three Ruff-specific rules about ambiguous Unicode characters are ignored because the site's text uses real em dashes, en dashes and Danish letters on purpose.

The pre-commit configuration runs `ruff check` and `ruff format --check` on any Python file under `scripts/` that a commit touches, and the project's fix task runs the same two commands in their fixing form. Beside Ruff sits basedpyright, a type checker, with a comment that marks the boundary between them: a linter does not follow a value across a function boundary, and on its first run the type checker found two real defects that Ruff had passed clean.

Where it goes wrong

Most trouble with Ruff comes from a configuration that is more literal than the people who wrote it.

An upgrade fails a branch nobody touched. Code that passed yesterday reports new violations after Ruff is updated. The usual cause is `select = ["ALL"]`, which takes on every rule a release adds, together with a version that floats. Pin the Ruff version so that an upgrade is a reviewed commit, and list the families the project wants instead of `ALL`, so a new family arrives only when someone adds it.

One folder stops following the project's rules. Violations the root configuration forbids pass in a subdirectory, or a rule the team switched off fires there. A `ruff.toml`, or a `pyproject.toml` with a `[tool.ruff]` table, sits in that folder, and because Ruff uses the closest file whole, every root setting stops applying below it. `ruff check --show-settings` on a file there prints the settings path it resolved to. Delete the nested file, or make it inherit with `extend = "../pyproject.toml"`.

A suppression outlives its reason. A `# noqa` comment went in for a good reason, the code changed, and the comment now hides nothing, or the next problem on that line. A bare `# noqa` is worse: it silences every rule on the line. Select RUF100, which reports `noqa` comments that no longer suppress anything, and PGH004, which asks for a specific code, and let `--fix` remove the dead ones. Here the import has since come into use, so the comment protects nothing and RUF100 reports it:

```
import json  # noqa: F401

print(json.dumps({"ok": True}))
```

The formatter and the linter disagree. `ruff format` rewrites a file and `ruff check` then flags the result. Some lint rules police what the formatter decides, such as trailing commas (COM812), indentation (W191, E111) and quote style (the Q rules), and the formatter wraps lines to the line length only as far as it can, so E501 can still fire on formatted code. Read the warnings `ruff format` prints, since a run free of them is the docs' own test that the two agree, ignore any rule a warning names, and treat E501 as a cue to restructure the line.

A bulk fix changes behaviour. After `ruff check --fix --unsafe-fixes` across the tree, tests fail or an error handler stops catching what it used to. Unsafe fixes are allowed to change runtime behaviour or drop comments, and the docs' own example changes which exception a piece of code raises. Keep `--unsafe-fixes` out of hooks and scripts, apply unsafe fixes one rule at a time with `--select` and read the diff, and promote a rule with `extend-safe-fixes` only once the team has checked it.

Further reading

- Ruff — what Ruff replaces, how fast it claims to be and who maintains it.

- The Ruff Linter — rule selection and precedence, safe and unsafe fixes, `noqa` comments and exit codes.
- The Ruff Formatter — Black compatibility and the lint rules that conflict with formatting.
- Rules — every rule by code, its origin and whether it can be fixed.
- Configuring Ruff — file discovery, the closest-file rule, `extend` and the command-line options.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Ruff?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/ruff/