Engineer

Ruff

🖌 Schneller Linter und Formatierer für Python

Was es ist

Ruff ist ein einziges, in Rust geschriebenes Kommandozeilenprogramm. Es prüft Python-Quelltext gegen einen Katalog von Lint-Regeln und schreibt dessen Layout in einen einheitlichen Stil um. Dahinter steht Astral, das Unternehmen hinter uv und ty. Zwei Unterbefehle erledigen die tägliche Arbeit: `ruff check` ist der Linter, `ruff format` der Formatierer. Ruff gibt an, 10- bis 100-mal schneller als Flake8 und Black zu laufen. Ein Cache verhindert, dass unveränderte Dateien erneut analysiert werden. Das Programm kann Flake8 mit Dutzenden seiner Plugins, Black, isort, pydocstyle, pyupgrade und autoflake ersetzen.

Der Linter enthält mehr als 900 Regeln, viele davon native Neuimplementierungen beliebter Flake8-Plugins. Jede Regel hat einen Code aus einem kurzen Präfix und drei Ziffern. Das Präfix nennt ihre Herkunft: F401, ein ungenutzter Import, kommt aus Pyflakes; E501, eine zu lange Zeile, aus pycodestyle. Die Konfiguration wählt Regeln einzeln oder nach Präfix anhand dieser Codes aus. So aktiviert `select = ["E", "F"]` jede Regel, deren Code mit E oder F beginnt, und der besondere Selektor `ALL` aktiviert alles. Viele Regeln haben eine Korrektur, eingeteilt in sichere, die das Laufzeitverhalten erhalten sollen, und unsichere, die es verändern können. `ruff check --fix` wendet nur die sicheren an, sofern nicht auch `--unsafe-fixes` übergeben wird.

Der Formatierer ist als direkt einsetzbarer Ersatz für Black gedacht. Bei Codebasen wie Django und Zulip entstehen mehr als 99,9 % identische Zeilen. Die Dokumentation sagt allerdings klar, dass die beiden nicht abwechselnd auf demselben Code eingesetzt werden sollen. Wie Black überspringt Ruff Bereiche zwischen `# fmt: off` und `# fmt: on` .

Beide Teile lesen dieselbe Konfiguration. Sie steht unter `[tool.ruff]` in `pyproject.toml` oder in einer Datei `ruff.toml` beziehungsweise `.ruff.toml`. Jede Python-Datei wird gegen die nächstgelegene Konfigurationsdatei in einem übergeordneten Verzeichnis geprüft, und diese wird vollständig verwendet: Ruff übernimmt nichts aus Elternverzeichnissen, solange die nähere Datei ihre übergeordnete Konfiguration nicht mit `extend` benennt. Optionen auf der Kommandozeile haben Vorrang vor den Einstellungen jeder Datei. Ohne eigene Einstellung dürfen Zeilen wie bei Black 88 Zeichen lang sein. Wenn keine Konfiguration eine Python-Version festlegt, nehmen die Regeln diese aus dem Feld `requires-python` des Projekts.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Ruff?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ein Team führt eine Idee von der Skizze bis zu einem Produkt, bei dem Ihre Kunden sich anmelden können, statt sie zwischen drei Dienstleistern weiterzureichen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. GIS-Kartenprodukt zu 10-fachem Umsatz geführt — 10x Umsatzwachstum durch die Leitung des Kartenprodukts

GIS-Kartenprodukt zu 10-fachem Umsatz geführt Kontakt aufnehmen →

Warum es wichtig ist

Geschwindigkeit entscheidet darüber, wo eine Prüfung laufen darf. Ein Linter, der so langsam ist, dass er die Arbeit unterbricht, wird in die Continuous Integration abgeschoben. Dort meldet er sich erst, wenn der Autor längst an etwas anderem arbeitet. Einer, der schnell antwortet, kann im Editor und in einem Commit-Hook laufen, während die Änderung noch frisch ist.

Die Bündelung beseitigt Konflikte zwischen Werkzeugen. Ein Projekt, das mit Flake8 und Plugins prüft, mit isort Imports sortiert und mit Black formatiert, muss mehrere Konfigurationen und festgeschriebene Versionen aufeinander abstimmen. Ruff bietet eine Konfiguration, eine festzuschreibende Version und eine Nummerierung der Regeln. Außerdem veröffentlicht es die Liste der Lint-Regeln, die mit dem eigenen Formatierer kollidieren. So lassen sich beide bewusst aufeinander abstimmen.

Die Exitcodes machen es ohne zusätzliches Steuerprogramm zu einer verbindlichen Prüfung. `ruff check` endet mit 0, wenn keine Verstöße übrig sind oder alle behoben wurden, mit 1 bei verbleibenden Verstößen und mit 2 bei

ungültiger Konfiguration oder Kommandozeile sowie bei einem Fehler in Ruff selbst, wie ESLint und Prettier. `ruff format --check` endet mit 1, sobald eine Datei neu formatiert würde. Eine Regel ist entweder ausgewählt oder ausgeschaltet. Jeder unkorrigierte Verstoß gegen eine ausgewählte Regel setzt den Exitcode auf 1.

Der Preis ist, dass jemand die Verantwortung für die Regelauswahl übernehmen muss. Ruff selbst rät dazu, Regeln mit `select` ausdrücklich aufzulisten, statt `extend-select` zu stapeln, mit E und F zu beginnen und jeweils eine Gruppe hinzuzunehmen. `ALL` sollte mit Bedacht eingesetzt werden, weil es bei jedem Upgrade neue Regeln aktiviert.

Wie es in der Praxis aussieht

Ruff übernahm Python bei einem für sechs Sprachen durchgesetzten Qualitätsstandard ohne Warnungen. Jede Sprache hatte ihren eigenen Linter. Jede Diagnose zählte als Fehler, und Inline-Unterdrückungen waren verboten. Für Ruff schließt das den Kommentar `# noqa` aus: Ein Python-Befund durfte ausschließlich im Code behoben werden. Pre-Commit- und Pre-Push-Hooks führten Linter und Tests aus, sodass die Prüfung lief, bevor ein Commit entstand.

Zum dokumentierten Werkzeugstand vom Oktober 2026 bekam Ruff auf dieser Website eine kleinere Aufgabe: die 22 Hilfsskripte neben dem Hugo-Build. Die Credits-Seite nennt Ruff unter den Werkzeugen, die den Quelltext prüfen. Die Regelauswahl in `pyproject.toml` wird durch Messungen bestimmt. Am 2026-10-15 wurden alle 54 Ruff-Regelfamilien auf diese Skripte angewendet. 39 meldeten nichts. Die ohne Befund gebliebenen Familien, deren Regeln zu diesem Code passen, wurden zu den ursprünglichen elf hinzugefügt, womit die Auswahl 33 umfasst. Die Sicherheitsfamilie S bleibt eingeschaltet. Fünf Regeln werden anhand ihrer Codes ignoriert, jeweils neben einem Kommentar, der erklärt, warum das beanstandete Muster hier die sichere Variante eines gefährlichen ist. Die übrigen Regeln der Familie bleiben dadurch verbindlich. Drei Ruff-eigene Regeln zu mehrdeutigen Unicode-Zeichen werden ignoriert, weil die Texte der Website echte Geviertstriche, Halbgeviertstriche und dänische Buchstaben absichtlich verwenden.

Die Pre-Commit-Konfiguration führt `ruff check` und `ruff format --check` für jede von einem Commit berührte Python-Datei unter `scripts/` aus. Die Korrekturaufgabe des Projekts führt dieselben beiden Befehle in ihrer korrigierenden Form aus. Neben Ruff steht basedpyright, ein Typprüfer. Ein

Kommentar grenzt beide voneinander ab: Ein Linter verfolgt einen Wert nicht über eine Funktionsgrenze hinweg. Bei seinem ersten Lauf fand der Typprüfer zwei echte Defekte, die Ruff als fehlerfrei durchgelassen hatte.

Wo es schiefgeht

Die meisten Schwierigkeiten mit Ruff entstehen durch eine Konfiguration, die wörtlicher arbeitet als die Menschen, die sie geschrieben haben.

Ein Upgrade lässt einen unberührten Branch scheitern. Code, der gestern bestand, meldet nach einer Ruff-Aktualisierung neue Verstöße. Meist ist `select = ["ALL"]` zusammen mit einer nicht festgeschriebenen Version die Ursache. Damit kommen alle Regeln hinzu, die eine Veröffentlichung ergänzt. Schreiben Sie die Ruff-Version fest, damit ein Upgrade ein geprüfter Commit ist, und nennen Sie statt `ALL` die vom Projekt gewünschten Familien. Eine neue Familie kommt dann erst hinzu, wenn jemand sie ergänzt.

Ein Verzeichnis folgt den Projektregeln nicht mehr. Verstöße, die die Wurzelkonfiguration verbietet, bleiben in einem Unterverzeichnis unbeanstandet, oder eine vom Team abgeschaltete Regel schlägt dort an. In diesem Verzeichnis liegt eine `ruff.toml` oder eine `pyproject.toml` mit einer Tabelle `[tool.ruff]`. Weil Ruff die nächstgelegene Datei vollständig verwendet, gilt darunter keine Einstellung der Wurzel mehr. `ruff check --show-settings` für eine dortige Datei gibt den ermittelten Konfigurationspfad aus. Löschen Sie die verschachtelte Datei oder lassen Sie sie mit `extend = "../pyproject.toml"` erben.

Eine Unterdrückung überlebt ihren Anlass. Ein Kommentar `# noqa` wurde aus gutem Grund ergänzt. Der Code änderte sich, und jetzt verdeckt der Kommentar nichts oder das nächste Problem auf dieser Zeile. Ein bloßes `# noqa` ist schlimmer: Es unterdrückt jede Regel auf der Zeile. Wählen Sie die Regel RUF100, die `noqa`-Kommentare meldet, die nichts mehr unterdrücken, und PGH004, die einen konkreten Code verlangt. Lassen Sie `--fix` die toten Kommentare entfernen. Hier wird der Import inzwischen verwendet. Der Kommentar schützt also nichts mehr, und RUF100 meldet ihn:

```
import json  # noqa: F401

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

Formatierer und Linter widersprechen sich. `ruff format` schreibt eine Datei um, und `ruff check` beanstandet anschließend das Ergebnis. Manche Lint-Regeln kontrollieren Entscheidungen des Formatierers: nachgestellte Kommata (COM812), Einrückung (W191, E111) und Anführungszeichen (die Q-Regeln). Außerdem bricht der Formatierer Zeilen nur so weit wie möglich auf die vorgesehene Länge um. E501 kann deshalb auch bei formatiertem Code anschlagen. Lesen Sie die Warnungen von `ruff format` : Laut Dokumentation ist ein warnungsfreier Lauf die Prüfung dafür, dass beide zusammenpassen. Ignorieren Sie jede in einer Warnung genannte Regel und verstehen Sie E501 als Anlass, die Zeile anders zu strukturieren.

Eine Massenkorrektur ändert das Verhalten. Nach `ruff check --fix --unsafe-fixes` über den ganzen Baum scheitern Tests, oder eine Fehlerbehandlung fängt nicht mehr ab, was sie zuvor abgefangen hat. Unsichere Korrekturen dürfen das Laufzeitverhalten verändern oder Kommentare entfernen. Das Beispiel der Dokumentation ändert, welche Ausnahme ein Stück Code auslöst. Halten Sie `--unsafe-fixes` aus Hooks und Skripten heraus, wenden Sie unsichere Korrekturen mit `--select` jeweils für eine Regel an und lesen Sie den Diff. Stufen Sie eine Regel mit `extend-safe-fixes` erst als sicher ein, wenn das Team sie geprüft hat.

Weiterführende Lektüre

- Ruff — Was Ruff ersetzt, wie schnell es nach eigener Aussage ist und wer es pflegt.
- Der Ruff-Linter — Regelauswahl und Vorrang, sichere und unsichere Korrekturen, `noqa` -Kommentare und Exitcodes.
- Der Ruff-Formatierer — Black-Kompatibilität und Lint-Regeln, die mit der Formatierung kollidieren.
- Regeln — Jede Regel nach Code, Herkunft und möglicher Korrektur.
- Ruff konfigurieren — Dateisuche, die nächstgelegene Konfiguration, `extend` und Kommandozeilenoptionen.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Ruff?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/ruff/