Engineer

Ruff

🖌 en snabb linter och formaterare för Python

Vad det är

Ruff är ett enda kommandoradsprogram, skrivet i Rust, som kontrollerar källkod i Python mot en katalog med lintregler och skriver om dess layout till en och samma stil. Bakom det står Astral, företaget bakom uv och ty, och två av underkommandona gör det dagliga arbetet: `ruff check` är lintern och `ruff format` är formateraren. Projektet uppger att det är 10 till 100 gånger snabbare än Flake8 och Black, håller en cache så att oförändrade filer inte analyseras igen och kan ersätta Flake8 med dussintals av dess insticksmoduler, Black, isort, pydocstyle, pyupgrade och autoflake.

Lintern levereras med mer än 900 regler, många av dem inbyggda omskrivningar av populära insticksmoduler till Flake8. Varje regel har en kod som består av ett kort prefix och tre siffror, och prefixet säger var regeln kommer ifrån: F401, en oanvänd import, kommer från Pyflakes, och E501, en rad som är för lång, från pycodestyle. En konfiguration väljer regler efter de koderna, en i taget eller per prefix, så `select = ["E", "F"]` slår på varje regel vars kod börjar med E eller F, och specialväljaren `ALL` slår på allt. Många regler har en automatisk rättning, uppdelad i säkra rättningar, som är tänkta att bevara beteendet vid körning, och osäkra, som kan ändra det. `ruff check --fix` tillämpar bara den säkra sorten, om inte `--unsafe-fixes` också anges.

Formateraren är utformad som en direkt ersättare för Black. I kodbaser som Django och Zulip blir mer än 99,9 % av raderna identiska, men dokumentationen säger rakt ut att de två inte är tänkta att bytas fram och tillbaka på samma kod. Precis som Black hoppar den över områden som omges av `# fmt: off` och `# fmt: on`.

Båda halvorna läser samma konfiguration, som ligger under `[tool.ruff]` i `pyproject.toml` eller i en fil som heter `ruff.toml` eller `.ruff.toml`. Varje Python-fil kontrolleras mot den närmaste konfigurationsfilen ovanför den, och den filen används i sin helhet: Ruff slår inte ihop något från överordnade kataloger om inte den närmare filen pekar ut sin förälder med `extend`. Flaggor på kommandoraden går före alla filer. Som standard får rader vara upp till 88 tecken långa, som i Black, och den Python-version som reglerna utgår från hämtas från projektets fält `requires-python` när ingen konfiguration anger någon.

Använd begreppet i ett beslut

Använd ”Vad är Ruff?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Ett team som tar en idé från en skiss till något dina kunder kan logga in i, i stället för att lämna den vidare mellan tre leverantörer.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Ledde GIS-kartappen till 10 gånger högre intäkter — 10x intäktstillväxt, med kartprodukten i spetsen

Ledde GIS-kartappen till 10 gånger högre intäkter Hör av er →

Varför det spelar roll

Hastigheten avgör var en kontroll får köras. En linter som är så långsam att den avbryter någon skjuts ut till CI, där den rapporterar först när författaren redan har gått vidare. En som svarar snabbt kan köras i editorn och i en hook vid varje commit, medan ändringen fortfarande är färsk.

Att samla allt i ett verktyg tar bort tvister mellan verktyg. Ett projekt som lintar med Flake8 och insticksmoduler, sorterar importer med isort och formaterar med Black måste hålla flera konfigurationer och versionslåsningar överens. Ruff erbjuder en konfiguration, en version att låsa och en numrering av regler, och projektet publicerar listan över de lintregler som krockar med dess egen formaterare, så att de två kan fås att stämma överens med avsikt.

Slutkoderna gör Ruff till en kvalitetsgrind utan något omslag. `ruff check` avslutas med 0 när inga överträdelser återstår eller när varje överträdelse har rättats, med 1 när överträdelser finns kvar, och med 2 när konfigurationen eller kommandoraden är ogiltig eller när Ruff självt fallerar, precis som ESLint och Prettier. `ruff format --check` avslutas med 1 när någon fil skulle formateras om. En regel är antingen vald eller avslagen, och varje vald regel som slår till sätter slutkoden till 1.

Kostnaden är att regeluppsättningen är ett beslut som någon måste äga. Ruffs eget råd är att lista regler uttryckligen med `select` i stället för att stapla `extend-select`, att börja med E och F och lägga till en grupp i taget, och att använda `ALL` med omdöme, eftersom den slår på nya regler vid varje uppgradering.

Så ser det ut i praktiken

Ruff täckte Python när ribban sattes vid noll varningar i sex språk, där varje språk hade en egen linter. Varje diagnostikmeddelande räknades som ett fel, och undertryckningar direkt i koden var förbjudna, vilket för Ruff utesluter kommentaren `# noqa`: ett fynd i Python kunde åtgärdas i koden och på inget annat sätt. Hookar för pre-commit och pre-push körde linters och tester, så att kontrollerna gjordes innan en commit ens fanns.

I den dokumenterade verktygsuppsättningen från oktober 2026 hade Ruff ett mindre jobb på den här webbplatsen: de 22 hjälpskripten bredvid Hugo-bygget. Kolofonen räknar upp Ruff bland verktygen som ”håller källkoden ärlig”. Regeluppsättningen i `pyproject.toml` är uppmätt snarare än vald. Den 15 oktober 2026 kördes var och en av Ruffs 54 regelfamiljer mot de skripten, 39 av dem rapporterade ingenting, och de tysta familjer vars ämne är just den här koden lades till de ursprungliga elva, så att urvalet nu omfattar 33 familjer. Säkerhetsfamiljen, S, är fortfarande påslagen, med fem regler ignorerade efter kod, var och en bredvid en kommentar som förklarar varför det flaggade mönstret här är den säkra tvillingen till ett farligt, så att familjens övriga regler fortfarande spärrar. Tre Ruff-specifika regler om tvetydiga Unicode-tecken ignoreras, eftersom webbplatsens text med avsikt använder riktiga långa och korta tankstreck och danska bokstäver.

Pre-commit-konfigurationen kör `ruff check` och `ruff format --check` på varje Python-fil under `scripts/` som en commit rör, och projektets task för rättningar kör samma två kommandon i deras rättande form. Bredvid Ruff står basedpyright, en typkontrollerare, med en kommentar som markerar gränsen mellan dem: en linter följer inte ett värde över en funktionsgräns, och vid sin första körning hittade typkontrolleraren två verkliga defekter som Ruff hade släppt igenom utan anmärkning.

Var det går fel

Det mesta besväret med Ruff kommer av en konfiguration som är mer bokstavlig än de människor som skrev den.

En gren som ingen har rört fallerar efter en uppgradering. Kod som klarade sig i går rapporterar nya överträdelser efter att Ruff har uppdaterats. Den vanliga orsaken är `select = ["ALL"]`, som tar in varje regel som en ny release lägger till, i kombination med en version som inte är låst. Lås Ruffs version så att en uppgradering blir en granskad commit, och lista de familjer som projektet vill ha i stället för `ALL`, så att en ny familj bara kommer in när någon lägger till den.

En mapp slutar följa projektets regler. Överträdelser som rotkonfigurationen förbjuder går igenom i en underkatalog, eller så slår en regel som teamet har stängt av till där. En `ruff.toml`, eller en `pyproject.toml` med en tabell `[tool.ruff]`, ligger i den mappen, och eftersom Ruff använder den närmaste filen i sin helhet slutar varje inställning från roten att gälla under den. `ruff check --show-settings` på en fil där skriver ut sökvägen till de inställningar som Ruff landade i. Ta bort den nästlade filen, eller låt den ärva med `extend = "../pyproject.toml"`.

En undertryckning överlever sitt skäl. En kommentar `# noqa` lades till av goda skäl, koden ändrades, och nu döljer kommentaren ingenting, eller döljer nästa problem på samma rad. En naken `# noqa` är värre: den tystar varje regel på raden. Slå på RUF100, som rapporterar `noqa`-kommentarer som inte längre undertrycker något, och PGH004, som kräver en specifik kod, och låt `--fix` ta bort de döda. Här har importen sedan dess kommit till användning, så kommentaren skyddar ingenting och RUF100 rapporterar den:

```
import json  # noqa: F401

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

Formateraren och lintern är oense. `ruff format` skriver om en fil, och `ruff check` flaggar sedan resultatet. Vissa lintregler vaktar det som formateraren bestämmer, som avslutande kommatecken (COM812), indrag (W191, E111) och stilen på citattecken (Q-reglerna), och formateraren försöker bara efter bästa förmåga hålla radlängden, så E501 kan fortfarande slå till på formaterad kod. Läs de varningar som `ruff format` skriver ut, eftersom en körning helt utan dem är dokumentationens eget test på att de två är överens, ignorera varje regel som en varning namnger och se E501 som en signal att strukturera om raden.

En massrättning ändrar beteendet. Efter `ruff check --fix --unsafe-fixes` över hela trädet fallerar testerna, eller så slutar en felhanterare att fånga det den brukade fånga. Osäkra rättningar får ändra beteendet vid körning eller ta bort kommentarer, och dokumentationens eget exempel ändrar vilket undantag en kodbit kastar. Håll `--unsafe-fixes` borta från hookar och skript, tillämpa osäkra rättningar en regel i taget med `--select` och läs diffen, och befordra en regel med `extend-safe-fixes` först när teamet har granskat den.

Läs mer

- Ruff — vad Ruff ersätter, hur snabbt projektet säger att det är och vem som underhåller det.
- The Ruff Linter — val av regler och deras företräde, säkra och osäkra rättningar, `noqa` -kommentarer och slutkoder.
- The Ruff Formatter — kompatibiliteten med Black och de lintregler som krockar med formateringen.
- Rules — varje regel efter kod, dess ursprung och om den går att rätta automatiskt.
- Configuring Ruff — hur konfigurationsfiler hittas, regeln om den närmaste filen, `extend` och kommandoradens flaggor.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Ruff?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/ruff/