astral-sh/ruff-lsp

A Language Server Protocol implementation for Ruff.

ref: v0.0.57

License: Other

Stars: 1287

Forks: 46

This PDF was generated by gitprint.me

Top Contributors

charliermarsh (149)

dependabot[bot] (123)

zanieb (18)

dhruvmanila (16)

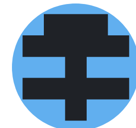
MichaReiser (11)

yaegassy (3)

konstin (3)

rchl (2)

bluetech (2)



figsoda (2)

mrKazzila (1)

MrGreenTea (1)

LDAP (1)

luccahuguet (1)

markis (1)

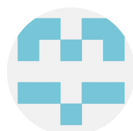
PushUpek (1)

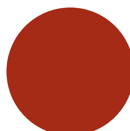
patillacode (1)

PedramNavid (1)

SauravMaheshkar (1)

diegorodriguezv (1)





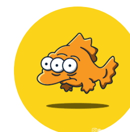
aspizu (1) igorlfs (1) WhyNotHugo (1) fannheyward (1) harupy (1)

FrancescElies (1)

vEnhance (1)

rockerBOO (1)

polyzen (1)



azmovi (1)

Chapter 0.0.0

root

README.md

ruff-lsp

[!NOTE]

As of Ruff v0.4.5, Ruff ships with a built-in language server written in Rust: ⚡ ruff server ⚡

ruff server supports the same feature set as ruff-lsp, but with superior performance and no installation required. ruff server was marked as stable in Ruff v0.5.3.

See the documentation for more.

A Language Server Protocol implementation for Ruff, an extremely fast Python linter and code formatter, written in Rust.

Ruff can be used to replace Flake8 (plus dozens of plugins), Black, isort, pyupgrade, and more, all while executing tens or hundreds of times faster than any individual tool.

ruff-lsp enables Ruff to be used in any editor that supports the LSP, including Neovim, Sublime Text, Emacs and more. For Visual Studio Code, check out the Ruff VS Code extension.

ruff-lsp supports surfacing Ruff diagnostics and providing Code Actions to fix them, but is intended to be used alongside another Python LSP in order to support features like navigation and autocompletion.

Highlights

“Quick Fix” actions for auto-fixable violations (like unused imports)

“Fix all”: automatically fix all auto-fixable violations

“Format Document”: Black-compatible code formatting

“Organize Imports”: isort-compatible import sorting

Installation

ruff-lsp is available as ruff-lsp on PyPI:

```
pip install ruff-lsp
```

Community packages

An Alpine Linux package is available in the testing repository:

```
apk add ruff-lsp
```

An Arch Linux package is available in the Extra repository:

```
pacman -S ruff-lsp
```

Setup

Once installed, ruff-lsp can be used with any editor that supports the Language Server Protocol, including Neovim, Emacs, Sublime Text, and more.

Example: Neovim

To use ruff-lsp with Neovim, follow these steps:

- Install ruff-lsp from PyPI along with nvim-lspconfig.
- Set up the Neovim LSP client using the suggested configuration (:h lspconfig-keybindings).
- Finally, configure ruff-lsp in your init.lua:

```
-- Configure `ruff-lsp`.
-- See: https://github.com/neovim/nvim-lspconfig/blob/master/doc/server_configurations.m
-- For the default config, along with instructions on how to customize the settings
require('lspconfig').ruff_lsp.setup {
  init_options = {
    settings = {
      -- Any extra CLI arguments for `ruff` go here.
      args = {},
    }
  }
}
```

Upon successful installation, you should see Ruff’s diagnostics surfaced directly in your editor:

Note that if you’re using Ruff alongside another LSP (like Pyright), you may want to defer to that LSP for certain capabilities, like textDocument/hover:

```
local on_attach = function(client, bufnr)
  if client.name == 'ruff_lsp' then
    -- Disable hover in favor of Pyright
    client.server_capabilities.hoverProvider = false
  end
end

require('lspconfig').ruff_lsp.setup {
  on_attach = on_attach,
}
```

And, if you’d like to use Ruff exclusively for linting, formatting, and organizing imports, you can disable those capabilities in Pyright:

```lua
require('lspconfig').pyright.setup {
  settings = {
    pyright = {
      -- Using Ruff's import organizer
      disableOrganizeImports = true,
    },
    python = {
      analysis = {
        -- Ignore all files for analysis to exclusively use Ruff for linting
        ignore = { '*' },
      },
    },
  },
}
```

Ruff also integrates with coc.nvim:

```json
{
    "languageserver": {
        "ruff-lsp": {
            "command": "ruff-lsp",
            "filetypes": [
                "python"
            ]
        }
    }
}
```

Example: Sublime Text

To use ruff-lsp with Sublime Text, install Sublime Text’s LSP and LSP-ruff package.

Upon successful installation, you should see errors surfaced directly in your editor:

Example: Helix

To use ruff-lsp with Helix, add something like the following to ~/.config/helix/languages.toml (in this case, with auto-format enabled):

```toml
[language-server.ruff]
command = "ruff-lsp"

[[language]]
name = "python"
language-servers = [ "ruff" ]
auto-format = true
```

Upon successful installation, you should see errors surfaced directly in your editor:

As of v23.10, Helix supports the use of multiple language servers for a given language. This enables, for example, the use of ruff-lsp alongside a language server like pyright:

```toml
[[language]]
name = "python"
language-servers = [ "pyright", "ruff" ]

[language-server.ruff]
command = "ruff-lsp"

[language-server.ruff.config.settings]
args = ["--ignore", "E501"]
```

Example: Lapce

To use ruff-lsp with Lapce, install the lapce-ruff-lsp plugin (which wraps ruff-lsp) from the Lapce plugins panel.

Upon successful installation, you should see errors surfaced directly in your editor:

Example: Kate

To use ruff-lsp with Kate, add something like the following to the LSP client's settings.json:

```
{
  "servers": {
    "python": {
      "command": ["ruff-lsp"],
      "url": "https://github.com/astral-sh/ruff-lsp",
      "highlightingModeRegex": "^Python$"
    }
  }
}
```

Fix safety

Ruff's automatic fixes are labeled as "safe" and "unsafe". By default, the "Fix all" action will not apply unsafe fixes. However, unsafe fixes can be applied manually with the "Quick fix" action. Application of unsafe fixes when using "Fix all" can be enabled by setting unsafe-fixes = true in your Ruff configuration file or adding --unsafe-fixes flag to the "Lint args" setting.

See the Ruff fix docs for more details on how fix safety works.

Jupyter Notebook Support

ruff-lsp has support for Jupyter Notebooks via the Notebook Document Synchronization capabilities of the Language Server Protocol which were added in 3.17. This allows ruff-lsp to provide full support for all of the existing capabilities available for Python files in Jupyter Notebooks, including diagnostics, code actions, and formatting.

This requires clients, such as Visual Studio Code, to support the notebook-related capabilities. In addition to the editor support, it also requires Ruff version v0.1.3 or later.

Settings

The exact mechanism by which settings will be passed to ruff-lsp will vary by editor. However, the following settings are supported:

Settings	Default	Description
codeAction.disableRuleComment.enable	true	Whether to display Quick Fix actions to disable rules via noqa suppression comments.
codeAction.fixViolation.enable	true	Whether to display Quick Fix actions to autofix violations.
fixAll	true	Whether to register Ruff as capable of handling source.fixAll actions.
format.args	[]	Additional command-line arguments to pass to ruff format, e.g., "args": ["--config=/path/to/pyproject.toml"]. Supports a subset of Ruff's command-line arguments, ignoring those that are required to operate the LSP, like --force-exclude and --verbose.
ignoreStandardLibrary	true	Whether to ignore files that are inferred to be part of the Python standard library.

Settings	Default	Description
interpreter	[]	Path to a Python interpreter to use to run the linter server.
lint.args	[]	Additional command-line arguments to pass to ruff check, e.g., "args": ["--config=/path/to/pyproject.toml"]. Supports a subset of Ruff's command-line arguments, ignoring those that are required to operate the LSP, like --force-exclude and --verbose.
lint.enable	true	Whether to enable linting. Set to false to use Ruff exclusively as a formatter.
lint.run	onType	Run Ruff on every keystroke (onType) or on save (onSave).
logLevel	error	Sets the tracing level for the extension: error, warn, info, or debug.
organizeImports	true	Whether to register Ruff as capable of handling source.organizeImports actions.
path	[]	Path to a custom ruff executable, e.g., ["/path/to/ruff"].
showSyntaxErrors	true	Whether to show syntax error diagnostics. *New in Ruff v0.5.0*

Development

- Install just, or see the justfile for corresponding commands.
- Create and activate a virtual environment (e.g., python -m venv .venv && source .venv/bin/activate).
- Install development dependencies (just install). To run the test_format.py test, you need to install a custom ruff build with --features format, e.g. maturin develop --features format -m ../ruff/crates/ruff_cli/Cargo.toml.
- To automatically format the codebase, run: just fmt.
- To run lint and type checks, run: just check.
- To run tests, run: just test. This is just a wrapper around pytest, which you can use as usual.

Release

- Bump the version in ruff_lsp/__init__.py.
- Make sure you use Python 3.7 installed and as your default Python.
- Run python -m venv .venv to create a venv and activate it.
- Run python -m pip install pip-tools to install pip-tools.
- Run rm requirements.txt requirements-dev.txt and then just lock to update ruff.
- Create a new PR and merge it.
- Create a new Release, enter v0.0.x (where x is the new version) into the *Choose a tag* selector. Click *Generate release notes*, curate the release notes and publish the release.
- The Release workflow publishes the LSP to PyPI.

License

MIT

LICENSE

MIT License

Copyright (c) 2022 Charles Marsh

Permission is hereby granted, free of charge, to any person obtaining a copy

```
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

The externally maintained libraries from which parts of the Software is derived
are:

- vscode-python-tools-extension-template, licensed as follows:
  """
    MIT License

    # TODO: Copyright (c) Microsoft Corporation.

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE
  """
}
```

pyproject.toml

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "ruff-lsp"
dynamic = ["version"]
description = "A Language Server Protocol implementation for Ruff."
authors = [
  { name = "Charlie Marsh", email = "charlie.r.marsh@gmail.com" },
]
maintainers = [
```

```
  { name = "Charlie Marsh", email = "charlie.r.marsh@gmail.com" },
]
requires-python = ">=3.7"
license = "MIT"
keywords = ["ruff", "lsp", "language-server", "language-server-protocol", "python"]
classifiers = [
  "Development Status :: 5 - Production/Stable",
  "Intended Audience :: Developers",
  "License :: OSI Approved :: MIT License",
  "Programming Language :: Python :: 3",
  "Programming Language :: Python :: 3.7",
  "Programming Language :: Python :: 3.8",
  "Programming Language :: Python :: 3.9",
  "Programming Language :: Python :: 3.10",
  "Programming Language :: Python :: 3.11",
  "Programming Language :: Python :: 3.12",
  "Programming Language :: Python :: Implementation :: CPython",
  "Programming Language :: Python :: Implementation :: PyPy",
  "Topic :: Software Development :: Libraries :: Python Modules",
  "Topic :: Software Development :: Quality Assurance",
  "Topic :: Software Development :: Testing",
  "Topic :: Utilities",
]
urls = { repository = "https://github.com/astral-sh/ruff-lsp" }
dependencies = [
  "packaging>=23.1",
  "pygls>=1.1.0",
  "lsprotocol>=2023.0.0",
  "ruff>=0.0.274",
  "typing_extensions",
]

[project.optional-dependencies]
dev = [
  "mypy==1.4.1",
  "pip-tools>=6.13.0,<7.0.0",
  "pytest>=7.3.1,<8.0.0",
  "pytest-asyncio==0.21.2",
  "python-lsp-jsonrpc==1.0.0",
]

[project.scripts]
ruff-lsp = "ruff_lsp.__main__:main"

[tool.hatch.version]
path = "ruff_lsp/__init__.py"

[tool.ruff]
line-length = 88
target-version = "py37"

[tool.ruff.lint]
select = [
  "E",
  "F",
  "W",
  "Q",
  "UP",
  "I",
  "N",
  "T201",
  "T203",
]
```

```
[tool.mypy]
files = ["ruff_lsp", "tests"]
no_implicit_optional = true
check_untyped_defs = true

[tool.pytest.ini_options]
addopts = "--tb=short"

[[tool.mypy.overrides]]
module = [
  "debugpy.*",
  "lsprotocol.*",
  "pygls.*",
  "pylsp_jsonrpc.*",
]
ignore_missing_imports = true
}
```

requirements-dev.txt

```
#
# This file is autogenerated by pip-compile with Python 3.7
# by the following command:
#
#    pip-compile --extra=dev --generate-hashes --output-file=requirements-dev.txt --
resolver=backtracking pyproject.toml
#
attrs==24.2.0 \
    --hash=sha256:5cfb1b9148b5b086569baec03f20d7b6bf3bcacc9a42bebf87ffaaca362f6346 \
    --hash=sha256:81921eb96de3191c8258c199618104dd27ac608d9366f5e35d011eae1867ede2
    # via
    #   cattrs
    #   lsprotocol
build==1.1.1 \
    --hash=sha256:8ed0851ee76e6e38adce47e4bee3b51c771d86c64cf578d0c2245567ee200e73 \
    --hash=sha256:8eea65bb45b1aac2e734ba2cc8dad3a6d97d97901a395bd0ed3e7b46953d2a31
    # via pip-tools
cattrs==23.1.2 \
    --hash=sha256:b2bb14311ac17bed0d58785e5a60f022e5431aca3932e3fc5cc8ed8639de50a4 \
    --hash=sha256:db1c821b8c537382b2c7c66678c3790091ca0275ac486c76f3c8f3920e83c657
    # via lsprotocol
click==8.1.7 \
    --hash=sha256:ae74fb96c20a0277a1d615f1e4d73c8414f5a98db8b799a7931d1582f3390c28 \
    --hash=sha256:ca9853ad459e787e2192211578cc907e7594e294c7ccc834310722b41b9ca6de
    # via pip-tools
exceptiongroup==1.2.2 \
    --hash=sha256:3111b9d131c238bec2f8f516e123e14ba243563fb135d3fe885990585aa7795b \
    --hash=sha256:47c2edf7c6738fafb49fd34290706d1a1a2f4d1c6df275526b62cbb4aa5393cc
    # via
    #   cattrs
    #   pytest
importlib-metadata==6.7.0 \
    --hash=sha256:1aaf550d4f73e5d6783e7acb77aec43d49da8017410afae93822cc9cca98c4d4 \
    --hash=sha256:cb52082e659e97afc5dac71e79de97d8681de3aa07ff18578330904a9d18e5b5
    # via
    #   attrs
    #   build
    #   click
    #   pluggy
    #   pytest
```

```
iniconfig==2.0.0 \
    --hash=sha256:2d91e135bf72d31a410b17c16da610a82cb55f6b0477d1a902134b24a455b8b3 \
    --hash=sha256:b6a85871a79d2e3b22d2d1b94ac2824226a63c6b741c88f7ae975f18b6778374
    # via pytest
lsprotocol==2023.0.0 \
    --hash=sha256:c9d92e12a3f4ed9317d3068226592860aab5357d93cf5b2451dc244eee8f35f2 \
    --hash=sha256:e85fc87ee26c816adca9eb497bb3db1a7c79c477a11563626e712eaccf926a05
    # via
    #   pygls
    #   ruff-lsp (pyproject.toml)
mypy==1.4.1 \
    --hash=sha256:01fd2e9f85622d981fd9063bfaef1aed6e336eaacca00892cd2d82801ab7c042 \
    --hash=sha256:0dde1d180cd84f0624c5dcaaa89c89775550a675aff96b5848de78fb11adabcd \
    --hash=sha256:141dedfdbfe8a04142881ff30ce6e6653c9685b354876b12e4fe6c78598b45e2 \
    --hash=sha256:16f0db5b641ba159eff72cff08edc3875f2b62b2fa2bc24f68c1e7a4e8232d01 \
    --hash=sha256:190b6bab0302cec4e9e6767d3eb66085aef2a1cc98fe04936d8a42ed2ba77bb7 \
    --hash=sha256:2460a58faeea905aeb1b9b36f5065f2dc9a9c6e4c992a6499a2360c6c74ceca3 \
    --hash=sha256:34a9239d5b3502c17f07fd7c0b2ae6b7dd7d7f6af35fbb5072c6208e76295816 \
    --hash=sha256:43b592511672017f5b1a483527fd2684347fdffc041c9ef53428c8dc530f79a3 \
    --hash=sha256:43d24f6437925ce50139a310a64b2ab048cb2d3694c84c71c3f2a1626d8101dc \
    --hash=sha256:45d32cec14e7b97af848bddd97d85ea4f0db4d5a149ed9676caa4eb2f7402bb4 \
    --hash=sha256:470c969bb3f9a9efcedbadcd19a74ffb34a25f8e6b0e02dae7c0e71f8372f97b \
    --hash=sha256:566e72b0cd6598503e48ea610e0052d1b8168e60a46e0bfd34b3acf2d57f96a8 \
    --hash=sha256:5703097c4936bbb9e9bce41478c8d08edd2865e177dc4c52be759f81ee4dd26c \
    --hash=sha256:7549fbf655e5825d787bbc9ecf6028731973f78088fbca3a1f4145c39ef09462 \
    --hash=sha256:8207b7105829eca6f3d774f64a904190bb2231de91b8b186d21ffd98005f14a7 \
    --hash=sha256:8c4d8e89aa7de683e2056a581ce63c46a0c41e31bd2b6d34144e2c80f5ea53dc \
    --hash=sha256:98324ec3ecf12296e6422939e54763faedbfcc502ea4a4c38502082711867258 \
    --hash=sha256:9bbcd9ab8ea1f2e1c8031c21445b511442cc45c89951e49bbf852cbb70755b1b \
    --hash=sha256:9d40652cc4fe33871ad3338581dca3297ff5f2213d0df345bcfbde5162abf0c9 \
    --hash=sha256:a2746d69a8196698146a3dbe29104f9eb6a2a4d8a27878d92169a6c0b74435b6 \
    --hash=sha256:ae704dcfaa180ff7c4cfbad23e74321a2b774f92ca77fd94ce1049175a21c97f \
    --hash=sha256:bfdca17c36ae01a21274a3c387a63aa1aafe72bff976522886869ef131b937f1 \
    --hash=sha256:c482e1246726616088532b5e964e39765b6d1520791348e6c9dc3af25b233828 \
    --hash=sha256:ca637024ca67ab24a7fd6f65d280572c3794665eaf5edcc7e90a866544076878 \
    --hash=sha256:e02d700ec8d9b1859790c0475df4e4092c7bf3272a4fd2c9f33d87fac4427b8f \
    --hash=sha256:e5952d2d18b79f7dc25e62e014fe5a23eb1a3d2bc66318df8988a01b1a037c5b
    # via ruff-lsp (pyproject.toml)
mypy-extensions==1.0.0 \
    --hash=sha256:4392f6c0eb8a5668a69e23d168ffa70f0be9ccfd32b5cc2d26a34ae5b844552d \
    --hash=sha256:75dbf8955dc00442a438fc4d0666508a9a97b6bd41aa2f0ffe9d2f2725af0782
    # via mypy
packaging==24.0 \
    --hash=sha256:2ddfb553fdf02fb784c234c7ba6ccc288296ceabec964ad2eae3777778130bc5 \
    --hash=sha256:eb82c5e3e56209074766e6885bb04b8c38a0c015d0a30036ebe7ece34c9989e9
    # via
    #   build
    #   pytest
    #   ruff-lsp (pyproject.toml)
pip-tools==6.14.0 \
    --hash=sha256:06366be0e08d86b416407333e998b4d305d5bd925151b08942ed149380ba3e47 \
    --hash=sha256:c5ad042cd27c0b343b10db1db7f77a7d087beafbec59ae6df1bba4d3368dfe8c
    # via ruff-lsp (pyproject.toml)
pluggy==1.2.0 \
    --hash=sha256:c2fd55a7d7a3863cba1a013e4e2414658b1d07b6bc57b3919e0c63c9abb99849 \
    --hash=sha256:d12f0c4b579b15f5e054301bb226ee85eeeba08ffec228092f8defbaa3a4c4b3
    # via pytest
pygls==1.2.1 \
    --hash=sha256:04f9b9c115b622dcc346fb390289066565343d60245a424eca77cb429b911ed8 \
    --hash=sha256:7dcfcf12b6f15beb606afa46de2ed348b65a279c340ef2242a9a35c22eeafe94
    # via ruff-lsp (pyproject.toml)
pyproject-hooks==1.1.0 \
    --hash=sha256:4b37730834edbd6bd37f26ece6b44802fb1c1ee2ece0e54ddff8bfc06db86965 \
    --hash=sha256:7ceeefe9aec63a1064c18d939bdc3adf2d8aa1988a510afec15151578b232aa2
```

```
    # via build
pytest==7.4.4 \
    --hash=sha256:2cf0005922c6ace4a3e2ec8b4080eb0d9753fdc93107415332f50ce9e7994280 \
    --hash=sha256:b090cdf5ed60bf4c45261be03239c2c1c22df034fbffe691abe93cd80cea01d8
    # via
    #   pytest-asyncio
    #   ruff-lsp (pyproject.toml)
pytest-asyncio==0.21.2 \
    --hash=sha256:ab664c88bb7998f711d8039cacd4884da6430886ae8bbd4eded552ed2004f16b \
    --hash=sha256:d67738fc232b94b326b9d060750beb16e0074210b98dd8b58a5239fa2a154f45
    # via ruff-lsp (pyproject.toml)
python-lsp-jsonrpc==1.0.0 \
    --hash=sha256:079b143be64b0a378bdb21dff5e28a8c1393fe7e8a654ef068322d754e545fc7 \
    --hash=sha256:7bec170733db628d3506ea3a5288ff76aa33c70215ed223abdb0d95e957660bd
    # via ruff-lsp (pyproject.toml)
ruff==0.6.6 \
    --hash=sha256:0adb801771bc1f1b8cf4e0a6fdc30776e7c1894810ff3b344e50da82ef50eeb1 \
    --hash=sha256:0fc030b6fd14814d69ac0196396f6761921bd20831725c7361e1b8100b818034 \
    --hash=sha256:2653fc3b2a9315bd809725c88dd2446550099728d077a04191febb5ea79a4f79 \
    --hash=sha256:488f8e15c01ea9afb8c0ba35d55bd951f484d0c1b7c5fd746ce3c47ccdedce68 \
    --hash=sha256:4b4d32c137bc781c298964dd4e52f07d6f7d57c03eae97a72d97856844aa510a \
    --hash=sha256:515a698254c9c47bb84335281a170213b3ee5eb47feebe903e1be10087a167ce \
    --hash=sha256:59627e97364329e4eae7d86fa7980c10e2b129e2293d25c478ebcb861b3e3fd6 \
    --hash=sha256:69c546f412dfae8bb9cc4f27f0e45cdd554e42fecbb34f03312b93368e1cd0a6 \
    --hash=sha256:6bb1b4995775f1837ab70f26698dd73852bbb82e8f70b175d2713c0354fe9182 \
    --hash=sha256:704da526c1e137f38c8a067a4a975fe6834b9f8ba7dbc5fd7503d58148851b8f \
    --hash=sha256:94c3f78c3d32190aafbb6bc5410c96cfed0a88aadb49c3f852bbc2aa9783a7d8 \
    --hash=sha256:a4c0698cc780bcb2c61496cbd56b6a3ac0ad858c966652f7dbf4ceb029252fbe \
    --hash=sha256:aadf81ddc8ab5b62da7aae78a91ec933cbae9f8f1663ec0325dae2c364e4ad84 \
    --hash=sha256:aefb0bd15f1cfa4c9c227b6120573bb3d6c4ee3b29fb54a5ad58f03859bc43c6 \
    --hash=sha256:bb858cd9ce2d062503337c5b9784d7b583bcf9d1a43c4df6ccb5eab774fbafcb \
    --hash=sha256:e368aef0cc02ca3593eae2fb8186b81c9c2b3f39acaaa1108eb6b4d04617e61f \
    --hash=sha256:efeede5815a24104579a0f6320660536c5ffc1c91ae94f8c65659af915fb9de9 \
    --hash=sha256:f5bc5398457484fc0374425b43b030e4668ed4d2da8ee7fdda0e926c9f11ccfb
    # via ruff-lsp (pyproject.toml)
tomli==2.0.1 \
    --hash=sha256:939de3e7a6161af0c887ef91b7d41a53e7c5a1ca976325f429cb46ea9bc30ecc \
    --hash=sha256:de526c12914f0c550d15924c62d72abc48d6fe7364aa87328337a31007fe8a4f
    # via
    #   build
    #   mypy
    #   pip-tools
    #   pytest
typed-ast==1.5.5 \
    --hash=sha256:042eb665ff6bf020dd2243307d11ed626306b82812aba21836096d229fdc6a10 \
    --hash=sha256:045f9930a1550d9352464e5149710d56a2aed23a2ffe78946478f7b5416f1ede \
    --hash=sha256:0635900d16ae133cab3b26c607586131269f88266954eb04ec31535c9a12ef1e \
    --hash=sha256:118c1ce46ce58fda78503eae14b7664163aa735b620b64b5b725453696f2a35c \
    --hash=sha256:16f7313e0a08c7de57f2998c85e2a69a642e97cb32f87eb65fbfe88381a5e44d \
    --hash=sha256:1efebbbf4604ad1283e963e8915daa240cb4bf5067053cf2f0baadc4d4fb51b8 \
    --hash=sha256:2188bc33d85951ea4ddad55d2b35598b2709d122c11c75cffd529fbc9965508e \
    --hash=sha256:2b946ef8c04f77230489f75b4b5a4a6f24c078be4aed241cfabe9cbf4156e7e5 \
    --hash=sha256:335f22ccb244da2b5c296e6f96b06ee9bed46526db0de38d2f0e5a6597b81155 \
    --hash=sha256:381eed9c95484ceef5ced626355fdc0765ab51d8553fec08661dce654a935db4 \
    --hash=sha256:429ae404f69dc94b9361bb62291885894b7c6fb4640d561179548c849f8492ba \
    --hash=sha256:44f214394fc1af23ca6d4e9e744804d890045d1643dd7e8229951e0ef39429b5 \
    --hash=sha256:48074261a842acf825af1968cd912f6f21357316080ebaca5f19abbb11690c8a \
    --hash=sha256:4bc1efe0ce3ffb74784e06460f01a223ac1f6ab31c6bc0376a21184bf5aabe3b \
    --hash=sha256:57bfc3cf35a0f2fdf0a88a3044aafaec1d2f24d8ae8cd87c4f58d615fb5b6311 \
    --hash=sha256:597fc66b4162f959ee6a96b978c0435bd63791e31e4f410622d19f1686d5e769 \
    --hash=sha256:5f7a8c46a8b333f71abd61d7ab9255440d4a588f34a21f126bbfc95f6049e686 \
    --hash=sha256:5fe83a9a44c4ce67c796a1b466c270c1272e176603d5e06f6afbc101a572859d \
    --hash=sha256:61443214d9b4c660dcf4b5307f15c12cb30bdfe9588ce6158f4a005baeb167b2 \
    --hash=sha256:622e4a006472b05cf6ef7f9f2636edc51bda670b7bbffa18d26b255269d3d814 \
```

```
    --hash=sha256:6eb936d107e4d474940469e8ec5b380c9b329b5f08b78282d46baeebd3692dc9 \
    --hash=sha256:7f58fabdde8dcbe764cef5e1a7fcb440f2463c1bbbec1cf2a86ca7bc1f95184b \
    --hash=sha256:83509f9324011c9a39faaef0922c6f720f9623afe3fe220b6d0b15638247206b \
    --hash=sha256:8c524eb3024edcc04e288db9541fe1f438f82d281e591c548903d5b77ad1ddd4 \
    --hash=sha256:94282f7a354f36ef5dbce0ef3467ebf6a258e370ab33d5b40c249fa996e590dd \
    --hash=sha256:b445c2abfecab89a932b20bd8261488d574591173d07827c1eda32c457358b18 \
    --hash=sha256:be4919b808efa61101456e87f2d4c75b228f4e52618621c77f1ddcaae15904fa \
    --hash=sha256:bfd39a41c0ef6f31684daff53befddae608f9daf6957140228a08e51f312d7e6 \
    --hash=sha256:c631da9710271cb67b08bd3f3813b7af7f4c69c319b75475436fcab8c3d21bee \
    --hash=sha256:cc95ffaaab2be3b25eb938779e43f513e0e538a84dd14a5d844b8f2932593d88 \
    --hash=sha256:d09d930c2d1d621f717bb217bf1fe2584616febb5138d9b3e8cdd26506c3f6d4 \
    --hash=sha256:d40c10326893ecab8a80a53039164a224984339b2c32a6baf55ecbd5b1df6431 \
    --hash=sha256:d41b7a686ce653e06c2609075d397ebd5b969d821b9797d029fccd71fdec8e04 \
    --hash=sha256:d5c0c112a74c0e5db2c75882a0adf3133adedcdbfd8cf7c9d6ed77365ab90a1d \
    --hash=sha256:e1a976ed4cc2d71bb073e1b2a250892a6e968ff02aa14c1f40eba4f365ffec02 \
    --hash=sha256:e48bf27022897577d8479eaed64701ecaf0467182448bd95759883300ca818c8 \
    --hash=sha256:ed4a1a42df8a3dfb6b40c3d2de109e935949f2f66b19703eafade03173f8f437 \
    --hash=sha256:f0aefdd66f1784c58f65b502b6cf8b121544680456d1cebbd300c2c813899274 \
    --hash=sha256:fc2b8c4e1bc5cd96c1a823a885e6b158f8451cf6f5530e1829390b4d27d0807f \
    --hash=sha256:fd946abf3c31fb50eee07451a6aedbfff912fcd13cf357363f5b4e834cc5e71a \
    --hash=sha256:fe58ef6a764de7b4b36edfc8592641f56e69b7163bba9f9c8089838ee596bfb2
    # via mypy
typing-extensions==4.7.1 \
    --hash=sha256:440d5dd3af93b060174bf433bccd69b0babc3b15b1a8dca43789fd7f61514b36 \
    --hash=sha256:b75ddc264f0ba5615db7ba217daeb99701ad295353c45f9e95963337ceeeffb2
    # via
    #   cattrs
    #   importlib-metadata
    #   mypy
    #   pytest-asyncio
    #   ruff-lsp (pyproject.toml)
ujson==5.7.0 \
    --hash=sha256:00343501dbaa5172e78ef0e37f9ebd08040110e11c12420ff7c1f9f0332d939e \
    --hash=sha256:0e4e8981c6e7e9e637e637ad8ffe948a09e5434bc5f52ecbb82b4b4cfc092bfb \
    --hash=sha256:0ee295761e1c6c30400641f0a20d381633d7622633cdf83a194f3c876a0e4b7e \
    --hash=sha256:137831d8a0db302fb6828ee21c67ad63ac537bddc4376e1aab1c8573756ee21c \
    --hash=sha256:14f9082669f90e18e64792b3fd0bf19f2b15e7fe467534a35ea4b53f3bf4b755 \
    --hash=sha256:16b2254a77b310f118717715259a196662baa6b1f63b1a642d12ab1ff998c3d7 \
    --hash=sha256:18679484e3bf9926342b1c43a3bd640f93a9eeeba19ef3d21993af7b0c44785d \
    --hash=sha256:24ad1aa7fc4e4caa41d3d343512ce68e41411fb92adf7f434a4d4b3749dc8f58 \
    --hash=sha256:26c2b32b489c393106e9cb68d0a02e1a7b9d05a07429d875c46b94ee8405bdb7 \
    --hash=sha256:2f242eec917bafdc3f73a1021617db85f9958df80f267db69c76d766058f7b19 \
    --hash=sha256:341f891d45dd3814d31764626c55d7ab3fd21af61fbc99d070e9c10c1190680b \
    --hash=sha256:35209cb2c13fcb9d76d249286105b4897b75a5e7f0efb0c0f4b90f222ce48910 \
    --hash=sha256:3d3b3499c55911f70d4e074c626acdb79a56f54262c3c83325ffb210fb03e44d \
    --hash=sha256:4a3d794afbf134df3056a813e5c8a935208cddeae975bd4bc0ef7e89c52f0ce0 \
    --hash=sha256:4c592eb91a5968058a561d358d0fef59099ed152cfb3e1cd14eee51a7a93879e \
    --hash=sha256:4ee997799a23227e2319a3f8817ce0b058923dbd31904761b788dc8f53bd3e30 \
    --hash=sha256:523ee146cdb2122bbd827f4dcc2a8e66607b3f665186bce9e4f78c9710b6d8ab \
    --hash=sha256:54384ce4920a6d35fa9ea8e580bc6d359e3eb961fa7e43f46c78e3ed162d56ff \
    --hash=sha256:5593263a7fcfb934107444bcfba9dde8145b282de0ee9f61e285e59a916dda0f \
    --hash=sha256:581c945b811a3d67c27566539bfcb9705ea09cb27c4be0002f7a553c8886b817 \
    --hash=sha256:5eba5e69e4361ac3a311cf44fa71bc619361b6e0626768a494771aacd1c2f09b \
    --hash=sha256:6411aea4c94a8e93c2baac096fbf697af35ba2b2ed410b8b360b3c0957a952d3 \
    --hash=sha256:64772a53f3c4b6122ed930ae145184ebaed38534c60f3d859d8c3f00911eb122 \
    --hash=sha256:67a19fd8e7d8cc58a169bea99fed5666023adf707a536d8f7b0a3c51dd498abf \
    --hash=sha256:6abb8e6d8f1ae72f0ed18287245f5b6d40094e2656d1eab6d99d666361514074 \
    --hash=sha256:6e80f0d03e7e8646fc3d79ed2d875cebd4c83846e129737fdc4c2532dbd43d9e \
    --hash=sha256:6faf46fa100b2b89e4db47206cf8a1ffb41542cdd34dde615b2fc2288954f194 \
    --hash=sha256:7312731c7826e6c99cdd3ac503cd9acd300598e7a80bcf41f604fee5f49f566c \
    --hash=sha256:75204a1dd7ec6158c8db85a2f14a68d2143503f4bafb9a00b63fe09d35762a5e \
    --hash=sha256:7592f40175c723c032cdbe9fe5165b3b5903604f774ab0849363386e99e1f253 \
    --hash=sha256:7b9dc5a90e2149643df7f23634fe202fed5ebc787a2a1be95cf23632b4d90651 \
    --hash=sha256:7df3fd35ebc14dafeea031038a99232b32f53fa4c3ecddb8bed132a43eefb8ad \
```

```
    --hash=sha256:800bf998e78dae655008dd10b22ca8dc93bdcfcc82f620d754a411592da4bbf2 \
    --hash=sha256:8b4257307e3662aa65e2644a277ca68783c5d51190ed9c49efebdd3cbfd5fa44 \
    --hash=sha256:90712dfc775b2c7a07d4d8e059dd58636bd6ff1776d79857776152e693bddea6 \
    --hash=sha256:9b0f2680ce8a70f77f5d70aaf3f013d53e6af6d7058727a35d8ceb4a71cdd4e9 \
    --hash=sha256:a5d2f44331cf04689eafac7a6596c71d6657967c07ac700b0ae1c921178645da \
    --hash=sha256:aae4d9e1b4c7b61780f0a006c897a4a1904f862fdab1abb3ea8f45bd11aa58f3 \
    --hash=sha256:adf445a49d9a97a5a4c9bb1d652a1528de09dd1c48b29f79f3d66cea9f826bf6 \
    --hash=sha256:af4639f684f425177d09ae409c07602c4096a6287027469157bfb6f83e01448b \
    --hash=sha256:afff311e9f065a8f03c3753db7011bae7beb73a66189c7ea5fcb0456b7041ea4 \
    --hash=sha256:b01a9af52a0d5c46b2c68e3f258fdef2eacaa0ce6ae3e9eb97983f5b1166edb6 \
    --hash=sha256:b522be14a28e6ac1cf818599aeff1004a28b42df4ed4d7bc819887b9dac915fc \
    --hash=sha256:b5ac3d5c5825e30b438ea92845380e812a476d6c2a1872b76026f2e9d8060fc2 \
    --hash=sha256:b6a6961fc48821d84b1198a09516e396d56551e910d489692126e90bf4887d29 \
    --hash=sha256:b7316d3edeba8a403686cdcad4af737b8415493101e7462a70ff73dd0609eafc \
    --hash=sha256:b738282e12a05f400b291966630a98d622da0938caa4bc93cf65adb5f4281c60 \
    --hash=sha256:bab10165db6a7994e67001733f7f2caf3400b3e11538409d8756bc9b1c64f7e8 \
    --hash=sha256:bea8d30e362180aafecabbdcbe0e1f0b32c9fa9e39c38e4af037b9d3ca36f50c \
    --hash=sha256:c0d1f7c3908357ee100aa64c4d1cf91edf99c40ac0069422a4fd5fd23b263263 \
    --hash=sha256:c3af9f9f22a67a8c9466a32115d9073c72a33ae627b11de6f592df0ee09b98b6 \
    --hash=sha256:c96e3b872bf883090ddf32cc41957edf819c5336ab0007d0cf3854e61841726d \
    --hash=sha256:cd90027e6d93e8982f7d0d23acf88c896d18deff1903dd96140613389b25c0dd \
    --hash=sha256:d2e43ccdba1cb5c6d3448eadf6fc0dae7be6c77e357a3abc968d1b44e265866d \
    --hash=sha256:d36a807a24c7d44f71686685ae6fbc8793d784bca1adf4c89f5f780b835b6243 \
    --hash=sha256:d7ff6ebb43bc81b057724e89550b13c9a30eda0f29c2f506f8b009895438f5a6 \
    --hash=sha256:d8cd622c069368d5074bd93817b31bdb02f8d818e57c29e206f10a1f9c6337dd \
    --hash=sha256:dda9aa4c33435147262cd2ea87c6b7a1ca83ba9b3933ff7df34e69fee9fced0c \
    --hash=sha256:e788e5d5dcae8f6118ac9b45d0b891a0d55f7ac480eddcb7f07263f2bcf37b23 \
    --hash=sha256:e87cec407ec004cf1b04c0ed7219a68c12860123dfb8902ef880d3d87a71c172 \
    --hash=sha256:ea7423d8a2f9e160c5e011119741682414c5b8dce4ae56590a966316a07a4618 \
    --hash=sha256:ed22f9665327a981f288a4f758a432824dc0314e4195a0eaeb0da56a477da94d \
    --hash=sha256:ed24406454bb5a31df18f0a423ae14beb27b28cdfa34f6268e7ebddf23da807e \
    --hash=sha256:f7f241488879d91a136b299e0c4ce091996c684a53775e63bb442d1a8e9ae22a \
    --hash=sha256:ff0004c3f5a9a6574689a553d1b7819d1a496b4f005a7451f339dc2d9f4cf98c
    # via python-lsp-jsonrpc
wheel==0.42.0 \
    --hash=sha256:177f9c9b0d45c47873b619f5b650346d632cdc35fb5e4d25058e09c9e581433d \
    --hash=sha256:c45be39f7882c9d34243236f2d63cbd58039e360f85d0913425fbd7ceea617a8
    # via pip-tools
zipp==3.15.0 \
    --hash=sha256:112929ad649da941c23de50f356a2b5570c954b65150642bccdd66bf194d224b \
    --hash=sha256:48904fc76a60e542af151aded95726c1a5c34ed43ab4134b597665c86d7ad556
    # via importlib-metadata

# WARNING: The following packages were not pinned, but pip requires them to be
# pinned when the requirements file includes hashes and the requirement is not
# satisfied by a package already installed. Consider using the --allow-unsafe flag.
# pip
# setuptools
}
```

requirements.txt

```
#
# This file is autogenerated by pip-compile with Python 3.7
# by the following command:
#
#    pip-compile --generate-hashes --output-file=requirements.txt --
resolver=backtracking pyproject.toml
#
attrs==24.2.0 \
```

```
    --hash=sha256:5cfb1b9148b5b086569baec03f20d7b6bf3bcacc9a42bebf87ffaaca362f6346 \
    --hash=sha256:81921eb96de3191c8258c199618104dd27ac608d9366f5e35d011eae1867ede2
    # via
    #   cattrs
    #   lsprotocol
cattrs==23.1.2 \
    --hash=sha256:b2bb14311ac17bed0d58785e5a60f022e5431aca3932e3fc5cc8ed8639de50a4 \
    --hash=sha256:db1c821b8c537382b2c7c66678c3790091ca0275ac486c76f3c8f3920e83c657
    # via lsprotocol
exceptiongroup==1.2.2 \
    --hash=sha256:3111b9d131c238bec2f8f516e123e14ba243563fb135d3fe885990585aa7795b \
    --hash=sha256:47c2edf7c6738fafb49fd34290706d1a1a2f4d1c6df275526b62cbb4aa5393cc
    # via cattrs
importlib-metadata==6.7.0 \
    --hash=sha256:1aaf550d4f73e5d6783e7acb77aec43d49da8017410afae93822cc9cca98c4d4 \
    --hash=sha256:cb52082e659e97afc5dac71e79de97d8681de3aa07ff18578330904a9d18e5b5
    # via attrs
lsprotocol==2023.0.0 \
    --hash=sha256:c9d92e12a3f4ed9317d3068226592860aab5357d93cf5b2451dc244eee8f35f2 \
    --hash=sha256:e85fc87ee26c816adca9eb497bb3db1a7c79c477a11563626e712eaccf926a05
    # via
    #   pygls
    #   ruff-lsp (pyproject.toml)
packaging==24.0 \
    --hash=sha256:2ddfb553fdf02fb784c234c7ba6ccc288296ceabec964ad2eae3777778130bc5 \
    --hash=sha256:eb82c5e3e56209074766e6885bb04b8c38a0c015d0a30036ebe7ece34c9989e9
    # via ruff-lsp (pyproject.toml)
pygls==1.2.1 \
    --hash=sha256:04f9b9c115b622dcc346fb390289066565343d60245a424eca77cb429b911ed8 \
    --hash=sha256:7dcfcf12b6f15beb606afa46de2ed348b65a279c340ef2242a9a35c22eeafe94
    # via ruff-lsp (pyproject.toml)
ruff==0.6.6 \
    --hash=sha256:0adb801771bc1f1b8cf4e0a6fdc30776e7c1894810ff3b344e50da82ef50eeb1 \
    --hash=sha256:0fc030b6fd14814d69ac0196396f6761921bd20831725c7361e1b8100b818034 \
    --hash=sha256:2653fc3b2a9315bd809725c88dd2446550099728d077a04191febb5ea79a4f79 \
    --hash=sha256:488f8e15c01ea9afb8c0ba35d55bd951f484d0c1b7c5fd746ce3c47ccdedce68 \
    --hash=sha256:4b4d32c137bc781c298964dd4e52f07d6f7d57c03eae97a72d97856844aa510a \
    --hash=sha256:515a698254c9c47bb84335281a170213b3ee5eb47feebe903e1be10087a167ce \
    --hash=sha256:59627e97364329e4eae7d86fa7980c10e2b129e2293d25c478ebcb861b3e3fd6 \
    --hash=sha256:69c546f412dfae8bb9cc4f27f0e45cdd554e42fecbb34f03312b93368e1cd0a6 \
    --hash=sha256:6bb1b4995775f1837ab70f26698dd73852bbb82e8f70b175d2713c0354fe9182 \
    --hash=sha256:704da526c1e137f38c8a067a4a975fe6834b9f8ba7dbc5fd7503d58148851b8f \
    --hash=sha256:94c3f78c3d32190aafbb6bc5410c96cfed0a88aadb49c3f852bbc2aa9783a7d8 \
    --hash=sha256:a4c0698cc780bcb2c61496cbd56b6a3ac0ad858c966652f7dbf4ceb029252fbe \
    --hash=sha256:aadf81ddc8ab5b62da7aae78a91ec933cbae9f8f1663ec0325dae2c364e4ad84 \
    --hash=sha256:aefb0bd15f1cfa4c9c227b6120573bb3d6c4ee3b29fb54a5ad58f03859bc43c6 \
    --hash=sha256:bb858cd9ce2d062503337c5b9784d7b583bcf9d1a43c4df6ccb5eab774fbafcb \
    --hash=sha256:e368aef0cc02ca3593eae2fb8186b81c9c2b3f39acaaa1108eb6b4d04617e61f \
    --hash=sha256:efeede5815a24104579a0f6320660536c5ffc1c91ae94f8c65659af915fb9de9 \
    --hash=sha256:f5bc5398457484fc0374425b43b030e4668ed4d2da8ee7fdda0e926c9f11ccfb
    # via ruff-lsp (pyproject.toml)
typing-extensions==4.7.1 \
    --hash=sha256:440d5dd3af93b060174bf433bccd69b0babc3b15b1a8dca43789fd7f61514b36 \
    --hash=sha256:b75ddc264f0ba5615db7ba217daeb99701ad295353c45f9e95963337ceeeffb2
    # via
    #   cattrs
    #   importlib-metadata
    #   ruff-lsp (pyproject.toml)
zipp==3.15.0 \
    --hash=sha256:112929ad649da941c23de50f356a2b5570c954b65150642bccdd66bf194d224b \
    --hash=sha256:48904fc76a60e542af151aded95726c1a5c34ed43ab4134b597665c86d7ad556
    # via importlib-metadata
}
```

Chapter 1.0.0

ruff_lsp

ruff_lsp/server.py

```
"""Implementation of the LSP server for Ruff."""

from __future__ import annotations

import asyncio
import enum
import json
import logging
import os
import re
import shutil
import sys
import sysconfig
from collections.abc import Iterable, Mapping
from dataclasses import dataclass
from pathlib import Path
from typing import Any, List, NamedTuple, Sequence, Union, cast

from lsprotocol.types import (
    CODE_ACTION_RESOLVE,
    INITIALIZE,
    NOTEBOOK_DOCUMENT_DID_CHANGE,
    NOTEBOOK_DOCUMENT_DID_CLOSE,
    NOTEBOOK_DOCUMENT_DID_OPEN,
    NOTEBOOK_DOCUMENT_DID_SAVE,
    TEXT_DOCUMENT_CODE_ACTION,
    TEXT_DOCUMENT_DID_CHANGE,
    TEXT_DOCUMENT_DID_CLOSE,
    TEXT_DOCUMENT_DID_OPEN,
    TEXT_DOCUMENT_DID_SAVE,
    TEXT_DOCUMENT_FORMATTING,
    TEXT_DOCUMENT_HOVER,
    TEXT_DOCUMENT_RANGE_FORMATTING,
    AnnotatedTextEdit,
    ClientCapabilities,
    CodeAction,
    CodeActionKind,
    CodeActionOptions,
    CodeActionParams,
    CodeDescription,
    Diagnostic,
    DiagnosticSeverity,
    DiagnosticTag,
    DidChangeNotebookDocumentParams,
    DidChangeTextDocumentParams,
    DidCloseNotebookDocumentParams,
    DidCloseTextDocumentParams,
    DidOpenNotebookDocumentParams,
    DidOpenTextDocumentParams,
```

```
    DidSaveNotebookDocumentParams,
    DidSaveTextDocumentParams,
    DocumentFormattingParams,
    DocumentRangeFormattingParams,
    DocumentRangeFormattingRegistrationOptions,
    Hover,
    HoverParams,
    InitializeParams,
    MarkupContent,
    MarkupKind,
    MessageType,
    NotebookCell,
    NotebookCellKind,
    NotebookDocument,
    NotebookDocumentSyncOptions,
    NotebookDocumentSyncOptionsNotebookSelectorType2,
    NotebookDocumentSyncOptionsNotebookSelectorType2CellsType,
    OptionalVersionedTextDocumentIdentifier,
    Position,
    PositionEncodingKind,
    Range,
    TextDocumentEdit,
    TextDocumentFilter_Type1,
    TextEdit,
    WorkspaceEdit,
)
from packaging.specifiers import SpecifierSet, Version
from pygls import server, uris, workspace
from pygls.workspace.position_codec import PositionCodec
from typing_extensions import Literal, Self, TypedDict, assert_never

from ruff_lsp import __version__, utils
from ruff_lsp.settings import (
    Run,
    UserSettings,
    WorkspaceSettings,
    lint_args,
    lint_enable,
    lint_run,
)
from ruff_lsp.utils import RunResult

logger = logging.getLogger(__name__)

RUFF_LSP_DEBUG = bool(os.environ.get("RUFF_LSP_DEBUG", False))

if RUFF_LSP_DEBUG:
    log_file = Path(__file__).parent.parent.joinpath("ruff-lsp.log")
    logging.basicConfig(filename=log_file, filemode="w", level=logging.DEBUG)
    logger.info("RUFF_LSP_DEBUG is active")


if sys.platform == "win32" and sys.version_info < (3, 8):
    # The ProactorEventLoop is required for subprocesses on Windows.
    # It's the default policy in Python 3.8, but not in Python 3.7.
    asyncio.set_event_loop_policy(asyncio.WindowsProactorEventLoopPolicy())


GLOBAL_SETTINGS: UserSettings = {}
WORKSPACE_SETTINGS: dict[str, WorkspaceSettings] = {}
INTERPRETER_PATHS: dict[str, str] = {}


class VersionModified(NamedTuple):
```

```
    version: Version
    """Last modified of the executable"""
    modified: float


EXECUTABLE_VERSIONS: dict[str, VersionModified] = {}
CLIENT_CAPABILITIES: dict[str, bool] = {
    CODE_ACTION_RESOLVE: True,
}

MAX_WORKERS = 5
LSP_SERVER = server.LanguageServer(
    name="Ruff",
    version=__version__,
    max_workers=MAX_WORKERS,
    notebook_document_sync=NotebookDocumentSyncOptions(
        notebook_selector=[
            NotebookDocumentSyncOptionsNotebookSelectorType2(
                cells=[
                    NotebookDocumentSyncOptionsNotebookSelectorType2CellsType(
                        language="python"
                    )
                ]
            )
        ],
        save=True,
    ),
)

TOOL_MODULE = "ruff.exe" if sys.platform == "win32" else "ruff"
TOOL_DISPLAY = "Ruff"

# Require at least Ruff v0.0.291 for formatting, but allow older versions for linting.
VERSION_REQUIREMENT_FORMATTER = SpecifierSet(">=0.0.291")
VERSION_REQUIREMENT_LINTER = SpecifierSet(">=0.0.189")
VERSION_REQUIREMENT_RANGE_FORMATTING = SpecifierSet(">=0.2.1")
# Version requirement for use of the `--output-format` option
VERSION_REQUIREMENT_OUTPUT_FORMAT = SpecifierSet(">=0.0.291")
# Version requirement after which Ruff avoids writing empty output for excluded files.
VERSION_REQUIREMENT_EMPTY_OUTPUT = SpecifierSet(">=0.1.6")

# Arguments provided to every Ruff invocation.
CHECK_ARGS = [
    "check",
    "--force-exclude",
    "--no-cache",
    "--no-fix",
    "--quiet",
    "--output-format",
    "json",
    "-",
]

# Arguments that are not allowed to be passed to `ruff check`.
UNSUPPORTED_CHECK_ARGS = [
    # Arguments that enforce required behavior. These can be ignored with a warning.
    "--force-exclude",
    "--no-cache",
    "--no-fix",
    "--quiet",
    # Arguments that contradict the required behavior. These can be ignored with a
    # warning.
    "--diff",
    "--exit-non-zero-on-fix",
```

```
    "-e",
    "--exit-zero",
    "--fix",
    "--fix-only",
    "-h",
    "--help",
    "--no-force-exclude",
    "--show-files",
    "--show-fixes",
    "--show-settings",
    "--show-source",
    "--silent",
    "--statistics",
    "--verbose",
    "-w",
    "--watch",
    # Arguments that are not supported at all, and will error when provided.
    # "--stdin-filename",
    # "--output-format",
]

# Arguments that are not allowed to be passed to `ruff format`.
UNSUPPORTED_FORMAT_ARGS = [
    # Arguments that enforce required behavior. These can be ignored with a warning.
    "--force-exclude",
    "--quiet",
    # Arguments that contradict the required behavior. These can be ignored with a
    # warning.
    "-h",
    "--help",
    "--no-force-exclude",
    "--silent",
    "--verbose",
    # Arguments that are not supported at all, and will error when provided.
    # "--stdin-filename",
]

# Standard code action kinds, scoped to Ruff.
SOURCE_FIX_ALL_RUFF = f"{CodeActionKind.SourceFixAll.value}.ruff"
SOURCE_ORGANIZE_IMPORTS_RUFF = f"{CodeActionKind.SourceOrganizeImports.value}.ruff"

# Notebook code action kinds.
NOTEBOOK_SOURCE_FIX_ALL = f"notebook.{CodeActionKind.SourceFixAll.value}"
NOTEBOOK_SOURCE_ORGANIZE_IMPORTS = (
    f"notebook.{CodeActionKind.SourceOrganizeImports.value}"
)

# Notebook code action kinds, scoped to Ruff.
NOTEBOOK_SOURCE_FIX_ALL_RUFF = f"notebook.{CodeActionKind.SourceFixAll.value}.ruff"
NOTEBOOK_SOURCE_ORGANIZE_IMPORTS_RUFF = (
    f"notebook.{CodeActionKind.SourceOrganizeImports.value}.ruff"
)


###
# Document
###


def _uri_to_fs_path(uri: str) -> str:
    """Convert a URI to a file system path."""
    path = uris.to_fs_path(uri)
    if path is None:
        # `pygls` raises a `Exception` as well in `workspace.TextDocument`.
```

```
        raise ValueError(f"Unable to convert URI to file path: {uri}")
    return path


@enum.unique
class DocumentKind(enum.Enum):
    """The kind of document."""

    Text = enum.auto()
    """A Python file."""

    Notebook = enum.auto()
    """A Notebook Document."""

    Cell = enum.auto()
    """A cell in a Notebook Document."""


@dataclass(frozen=True)
class Document:
    """A document representing either a Python file, a Notebook cell, or a Notebook."""

    uri: str
    path: str
    source: str
    kind: DocumentKind
    version: int | None

    @classmethod
    def from_text_document(cls, text_document: workspace.TextDocument) -> Self:
        """Create a `Document` from the given Text Document."""
        return cls(
            uri=text_document.uri,
            path=text_document.path,
            kind=DocumentKind.Text,
            source=text_document.source,
            version=text_document.version,
        )

    @classmethod
    def from_notebook_document(cls, notebook_document: NotebookDocument) -> Self:
        """Create a `Document` from the given Notebook Document."""
        return cls(
            uri=notebook_document.uri,
            path=_uri_to_fs_path(notebook_document.uri),
            kind=DocumentKind.Notebook,
            source=_create_notebook_json(notebook_document),
            version=notebook_document.version,
        )

    @classmethod
    def from_notebook_cell(cls, notebook_cell: NotebookCell) -> Self:
        """Create a `Document` from the given Notebook cell."""
        return cls(
            uri=notebook_cell.document,
            path=_uri_to_fs_path(notebook_cell.document),
            kind=DocumentKind.Cell,
            source=_create_single_cell_notebook_json(
                LSP_SERVER.workspace.get_text_document(notebook_cell.document).source
            ),
            version=None,
        )

    @classmethod
```

```
    def from_cell_or_text_uri(cls, uri: str) -> Self:
        """Create a `Document` representing either a Python file or a Notebook
cell from
        the given URI.

        The function will try to get the Notebook cell first, and if there's no cell
        with the given URI, it will fallback to the text document.
        """
        notebook_document = LSP_SERVER.workspace.get_notebook_document(cell_uri=uri)
        if notebook_document is not None:
            notebook_cell = next(
                (
                    notebook_cell
                    for notebook_cell in notebook_document.cells
                    if notebook_cell.document == uri
                ),
                None,
            )
            if notebook_cell is not None:
                return cls.from_notebook_cell(notebook_cell)

        # Fall back to the Text Document representing a Python file.
        text_document = LSP_SERVER.workspace.get_text_document(uri)
        return cls.from_text_document(text_document)

    @classmethod
    def from_uri(cls, uri: str) -> Self:
        """Create a `Document` representing either a Python file or a Notebook from
        the given URI.

        The URI can be a file URI, a notebook URI, or a cell URI. The function will
        try to get the notebook document first, and if there's no notebook document
        with the given URI, it will fallback to the text document.
        """
        # First, try to get the Notebook Document assuming the URI is a Cell URI.
        notebook_document = LSP_SERVER.workspace.get_notebook_document(cell_uri=uri)
        if notebook_document is None:
            # If that fails, try to get the Notebook Document assuming the URI is a
            # Notebook URI.
            notebook_document = LSP_SERVER.workspace.get_notebook_document(
                notebook_uri=uri
            )
        if notebook_document:
            return cls.from_notebook_document(notebook_document)

        # Fall back to the Text Document representing a Python file.
        text_document = LSP_SERVER.workspace.get_text_document(uri)
        return cls.from_text_document(text_document)

    def is_stdlib_file(self) -> bool:
        """Return True if the document belongs to standard library."""
        return utils.is_stdlib_file(self.path)


SourceValue = Union[str, List[str]]


class CodeCell(TypedDict):
    """A code cell in a Jupyter notebook."""

    cell_type: Literal["code"]
    metadata: Any
    outputs: list[Any]
    source: SourceValue
```

```python
class MarkdownCell(TypedDict):
    """A markdown cell in a Jupyter notebook."""

    cell_type: Literal["markdown"]
    metadata: Any
    source: SourceValue


class Notebook(TypedDict):
    """The JSON representation of a Notebook Document."""

    metadata: Any
    nbformat: int
    nbformat_minor: int
    cells: list[CodeCell | MarkdownCell]


def _create_notebook_json(notebook_document: NotebookDocument) -> str:
    """Create a JSON representation of the given Notebook Document."""
    cells: list[CodeCell | MarkdownCell] = []
    for notebook_cell in notebook_document.cells:
        cell_document = LSP_SERVER.workspace.get_text_document(notebook_cell.document)
        if notebook_cell.kind is NotebookCellKind.Code:
            cells.append(
                {
                    "cell_type": "code",
                    "metadata": {},
                    "outputs": [],
                    "source": cell_document.source,
                }
            )
        else:
            cells.append(
                {
                    "cell_type": "markdown",
                    "metadata": {},
                    "source": cell_document.source,
                }
            )
    return json.dumps(
        {
            "metadata": {},
            "nbformat": 4,
            "nbformat_minor": 5,
            "cells": cells,
        }
    )


def _create_single_cell_notebook_json(source: str) -> str:
    """Create a JSON representation of a single cell Notebook Document containing
    the given source."""
    return json.dumps(
        {
            "metadata": {},
            "nbformat": 4,
            "nbformat_minor": 5,
            "cells": [
                {
                    "cell_type": "code",
                    "metadata": {},
                    "outputs": [],
```

```
                    "source": source,
                }
            ],
        }
    )


###
# Linting.
###


@LSP_SERVER.feature(TEXT_DOCUMENT_DID_OPEN)
async def did_open(params: DidOpenTextDocumentParams) -> None:
    """LSP handler for textDocument/didOpen request."""
    document = Document.from_text_document(
        LSP_SERVER.workspace.get_text_document(params.text_document.uri)
    )
    settings = _get_settings_by_document(document.path)
    if not lint_enable(settings):
        return None

    diagnostics = await _lint_document_impl(document, settings)
    LSP_SERVER.publish_diagnostics(document.uri, diagnostics)


@LSP_SERVER.feature(TEXT_DOCUMENT_DID_CLOSE)
def did_close(params: DidCloseTextDocumentParams) -> None:
    """LSP handler for textDocument/didClose request."""
    text_document = LSP_SERVER.workspace.get_text_document(params.text_document.uri)
    # Publishing empty diagnostics to clear the entries for this file.
    LSP_SERVER.publish_diagnostics(text_document.uri, [])


@LSP_SERVER.feature(TEXT_DOCUMENT_DID_SAVE)
async def did_save(params: DidSaveTextDocumentParams) -> None:
    """LSP handler for textDocument/didSave request."""
    text_document = LSP_SERVER.workspace.get_text_document(params.text_document.uri)
    settings = _get_settings_by_document(text_document.path)
    if not lint_enable(settings):
        return None

    if lint_run(settings) in (
        Run.OnType,
        Run.OnSave,
    ):
        document = Document.from_text_document(text_document)
        diagnostics = await _lint_document_impl(document, settings)
        LSP_SERVER.publish_diagnostics(document.uri, diagnostics)


@LSP_SERVER.feature(TEXT_DOCUMENT_DID_CHANGE)
async def did_change(params: DidChangeTextDocumentParams) -> None:
    """LSP handler for textDocument/didChange request."""
    text_document = LSP_SERVER.workspace.get_text_document(params.text_document.uri)
    settings = _get_settings_by_document(text_document.path)
    if not lint_enable(settings):
        return None

    if lint_run(settings) == Run.OnType:
        document = Document.from_text_document(text_document)
        diagnostics = await _lint_document_impl(document, settings)
        LSP_SERVER.publish_diagnostics(document.uri, diagnostics)
```

```python
@LSP_SERVER.feature(NOTEBOOK_DOCUMENT_DID_OPEN)
async def did_open_notebook(params: DidOpenNotebookDocumentParams) -> None:
    """LSP handler for notebookDocument/didOpen request."""
    notebook_document = LSP_SERVER.workspace.get_notebook_document(
        notebook_uri=params.notebook_document.uri
    )
    if notebook_document is None:
        log_warning(f"No notebook document found for {params.notebook_document.uri!r}")
        return None

    document = Document.from_notebook_document(notebook_document)
    settings = _get_settings_by_document(document.path)
    if not lint_enable(settings):
        return None

    diagnostics = await _lint_document_impl(document, settings)

    # Publish diagnostics for each cell.
    for cell_idx, diagnostics in _group_diagnostics_by_cell(diagnostics).items():
        LSP_SERVER.publish_diagnostics(
            # The cell indices are 1-based in Ruff.
            params.notebook_document.cells[cell_idx - 1].document,
            diagnostics,
        )


@LSP_SERVER.feature(NOTEBOOK_DOCUMENT_DID_CLOSE)
def did_close_notebook(params: DidCloseNotebookDocumentParams) -> None:
    """LSP handler for notebookDocument/didClose request."""
    # Publishing empty diagnostics to clear the entries for all the cells in this
    # Notebook Document.
    for cell_text_document in params.cell_text_documents:
        LSP_SERVER.publish_diagnostics(cell_text_document.uri, [])


@LSP_SERVER.feature(NOTEBOOK_DOCUMENT_DID_SAVE)
async def did_save_notebook(params: DidSaveNotebookDocumentParams) -> None:
    """LSP handler for notebookDocument/didSave request."""
    await _did_change_or_save_notebook(
        params.notebook_document.uri, run_types=[Run.OnSave, Run.OnType]
    )


@LSP_SERVER.feature(NOTEBOOK_DOCUMENT_DID_CHANGE)
async def did_change_notebook(params: DidChangeNotebookDocumentParams) -> None:
    """LSP handler for notebookDocument/didChange request."""
    await _did_change_or_save_notebook(
        params.notebook_document.uri, run_types=[Run.OnType]
    )


def _group_diagnostics_by_cell(
    diagnostics: Iterable[Diagnostic],
) -> Mapping[int, list[Diagnostic]]:
    """Group diagnostics by cell index.

    The function will return a mapping from cell number to a list of diagnostics for
    that cell. The mapping will be empty if the diagnostic doesn't contain the cell
    information.
    """
    cell_diagnostics: dict[int, list[Diagnostic]] = {}
    for diagnostic in diagnostics:
        cell = cast(DiagnosticData, diagnostic.data).get("cell")
```

```
        if cell is not None:
            cell_diagnostics.setdefault(cell, []).append(diagnostic)
    return cell_diagnostics


async def _did_change_or_save_notebook(
    notebook_uri: str, *, run_types: Sequence[Run]
) -> None:
    """Handle notebookDocument/didChange and notebookDocument/didSave requests."""
    notebook_document = LSP_SERVER.workspace.get_notebook_document(
        notebook_uri=notebook_uri
    )
    if notebook_document is None:
        log_warning(f"No notebook document found for {notebook_uri!r}")
        return None

    document = Document.from_notebook_document(notebook_document)
    settings = _get_settings_by_document(document.path)
    if not lint_enable(settings):
        return None

    if lint_run(settings) in run_types:
        cell_diagnostics = _group_diagnostics_by_cell(
            await _lint_document_impl(document, settings)
        )

        # Publish diagnostics for every code cell, replacing the previous diagnostics.
        # This is required here because a cell containing diagnostics in the first run
        # might not contain any diagnostics in the second run. In that case, we need to
        # clear the diagnostics for that cell which is done by publishing empty
        # diagnostics.
        for cell_idx, cell in enumerate(notebook_document.cells):
            if cell.kind is not NotebookCellKind.Code:
                continue
            LSP_SERVER.publish_diagnostics(
                cell.document,
                # The cell indices are 1-based in Ruff.
                cell_diagnostics.get(cell_idx + 1, []),
            )


async def _lint_document_impl(
    document: Document, settings: WorkspaceSettings
) -> list[Diagnostic]:
    result = await _run_check_on_document(document, settings)
    if result is None:
        return []

    # For `ruff check`, 0 indicates successful completion with no remaining
    # diagnostics, 1 indicates successful completion with remaining diagnostics,
    # and 2 indicates an error.
    if result.exit_code not in (0, 1):
        if result.stderr:
            show_error(f"Ruff: Lint failed ({result.stderr.decode('utf-8')})")
        return []

    return (
        _parse_output(result.stdout, settings.get("showSyntaxErrors", True))
        if result.stdout
        else []
    )


def _parse_fix(content: Fix | LegacyFix | None) -> Fix | None:
```

```
    """Parse the fix from the Ruff output."""
    if content is None:
        return None

    if content.get("edits") is None:
        # Prior to v0.0.260, Ruff returned a single edit.
        legacy_fix = cast(LegacyFix, content)
        return {
            "applicability": None,
            "message": legacy_fix.get("message"),
            "edits": [
                {
                    "content": legacy_fix["content"],
                    "location": legacy_fix["location"],
                    "end_location": legacy_fix["end_location"],
                }
            ],
        }
    else:
        # Since v0.0.260, Ruff returns a list of edits.
        fix = cast(Fix, content)

        # Since v0.0.266, Ruff returns one based column indices
        if fix.get("applicability") is not None:
            for edit in fix["edits"]:
                edit["location"]["column"] = edit["location"]["column"] - 1
                edit["end_location"]["column"] = edit["end_location"]["column"] - 1

        return fix


def _parse_output(content: bytes, show_syntax_errors: bool) -> list[Diagnostic]:
    """Parse Ruff's JSON output."""
    diagnostics: list[Diagnostic] = []

    # Ruff's output looks like:
    # [
    #   {
    #     "cell": null,
    #     "code": "F841",
    #     "message": "Local variable `x` is assigned to but never used",
    #     "location": {
    #       "row": 2,
    #       "column": 5
    #     },
    #     "end_location": {
    #       "row": 2,
    #       "column": 6
    #     },
    #     "fix": {
    #       "applicability": "Unspecified",
    #       "message": "Remove assignment to unused variable `x`",
    #       "edits": [
    #         {
    #           "content": "",
    #           "location": {
    #             "row": 2,
    #             "column": 1
    #           },
    #           "end_location": {
    #             "row": 3,
    #             "column": 1
    #           }
    #         }
```

```
    #       ]
    #     },
    #     "filename": "/path/to/test.py",
    #     "noqa_row": 2
    #   },
    #   ...
    # ]
    #
    # Input:
    # ```python
    # def a():
    #     x = 0
    #     print()
    # ```
    #
    # Cell represents the cell number in a Notebook Document. It is null for normal
    # Python files.
    for check in json.loads(content):
        if not show_syntax_errors and check["code"] is None:
            continue
        start = Position(
            line=max([int(check["location"]["row"]) - 1, 0]),
            character=int(check["location"]["column"]) - 1,
        )
        end = Position(
            line=max([int(check["end_location"]["row"]) - 1, 0]),
            character=int(check["end_location"]["column"]) - 1,
        )
        diagnostic = Diagnostic(
            range=Range(start=start, end=end),
            message=check.get("message"),
            code=check["code"],
            code_description=_get_code_description(check.get("url")),
            severity=_get_severity(check["code"]),
            source=TOOL_DISPLAY,
            data=DiagnosticData(
                fix=_parse_fix(check.get("fix")),
                # Available since Ruff v0.0.253.
                noqa_row=check.get("noqa_row"),
                # Available since Ruff v0.1.0.
                cell=check.get("cell"),
            ),
            tags=_get_tags(check["code"]),
        )
        diagnostics.append(diagnostic)

    return diagnostics


def _get_code_description(url: str | None) -> CodeDescription | None:
    if url is None:
        return None
    else:
        return CodeDescription(href=url)


def _get_tags(code: str) -> list[DiagnosticTag] | None:
    if code in {
        "F401",  # `module` imported but unused
        "F841",  # local variable `name` is assigned to but never used
    }:
        return [DiagnosticTag.Unnecessary]
    return None
```

```python
def _get_severity(code: str) -> DiagnosticSeverity:
    if code in {
        "F821",  # undefined name `name`
        "E902",  # `IOError`
        "E999",  # `SyntaxError`
        None,  # `SyntaxError` as of Ruff v0.5.0
    }:
        return DiagnosticSeverity.Error
    else:
        return DiagnosticSeverity.Warning


NOQA_REGEX = re.compile(
    r"(?i:# (?:(?:ruff|flake8): )?(?P<noqa>noqa))"
    r"(?::\s?(?P<codes>([A-Z]+[0-9]+(?:[,\s]+)?)+))?"
)
CODE_REGEX = re.compile(r"[A-Z]+[0-9]+")


@LSP_SERVER.feature(TEXT_DOCUMENT_HOVER)
async def hover(params: HoverParams) -> Hover | None:
    """LSP handler for textDocument/hover request.

    This works for both Python files and Notebook Documents. For Notebook Documents,
    the hover works at the cell level.
    """
    document = LSP_SERVER.workspace.get_text_document(params.text_document.uri)
    match = NOQA_REGEX.search(document.lines[params.position.line])
    if not match:
        return None

    codes = match.group("codes")
    if not codes:
        return None

    codes_start = match.start("codes")
    for match in CODE_REGEX.finditer(codes):
        start, end = match.span()
        start += codes_start
        end += codes_start
        if start <= params.position.character < end:
            code = match.group()
            result = await _run_subcommand_on_document(
                document, VERSION_REQUIREMENT_LINTER, args=["--explain", code]
            )
            if result.stdout:
                return Hover(
                    contents=MarkupContent(
                        kind=MarkupKind.Markdown,
                        value=result.stdout.decode("utf-8").strip(),
                    )
                )

    return None


###
# Code Actions.
###


class TextDocument(TypedDict):
    uri: str
```

```python
    version: int


class Location(TypedDict):
    row: int
    column: int


class Edit(TypedDict):
    content: str
    location: Location
    end_location: Location


class Fix(TypedDict):
    """A fix for a diagnostic, represented as a list of edits."""

    applicability: str | None
    message: str | None
    edits: list[Edit]


class DiagnosticData(TypedDict, total=False):
    fix: Fix | None
    noqa_row: int | None
    cell: int | None


class LegacyFix(TypedDict):
    """A fix for a diagnostic, represented as a single edit.

    Matches Ruff's output prior to v0.0.260.
    """

    message: str | None
    content: str
    location: Location
    end_location: Location


@LSP_SERVER.feature(
    TEXT_DOCUMENT_CODE_ACTION,
    CodeActionOptions(
        code_action_kinds=[
            # Standard code action kinds.
            CodeActionKind.QuickFix,
            CodeActionKind.SourceFixAll,
            CodeActionKind.SourceOrganizeImports,
            # Standard code action kinds, scoped to Ruff.
            SOURCE_FIX_ALL_RUFF,
            SOURCE_ORGANIZE_IMPORTS_RUFF,
            # Notebook code action kinds.
            NOTEBOOK_SOURCE_FIX_ALL,
            NOTEBOOK_SOURCE_ORGANIZE_IMPORTS,
            # Notebook code action kinds, scoped to Ruff.
            NOTEBOOK_SOURCE_FIX_ALL_RUFF,
            NOTEBOOK_SOURCE_ORGANIZE_IMPORTS_RUFF,
        ],
        resolve_provider=True,
    ),
)
async def code_action(params: CodeActionParams) -> list[CodeAction] | None:
    """LSP handler for textDocument/codeAction request.
```

```
    Code actions work at a text document level which is either a Python file or a
    cell in a Notebook document. The function will try to get the Notebook cell
    first, and if there's no cell with the given URI, it will fallback to the text
    document.
    """

    def document_from_kind(uri: str, kind: str) -> Document:
        if kind in (
            # For `notebook`-scoped actions, use the Notebook Document instead of
            # the cell, despite being passed the URI of the first cell.
            # See: https://github.com/microsoft/vscode/issues/193120
            NOTEBOOK_SOURCE_FIX_ALL,
            NOTEBOOK_SOURCE_ORGANIZE_IMPORTS,
            NOTEBOOK_SOURCE_FIX_ALL_RUFF,
            NOTEBOOK_SOURCE_ORGANIZE_IMPORTS_RUFF,
        ):
            return Document.from_uri(uri)
        else:
            return Document.from_cell_or_text_uri(uri)

    document_path = _uri_to_fs_path(params.text_document.uri)
    settings = _get_settings_by_document(document_path)

    if settings.get("ignoreStandardLibrary", True) and utils.is_stdlib_file(
        document_path
    ):
        # Don't format standard library files.
        # Publishing empty list clears the entry.
        return None

    if settings["organizeImports"]:
        # Generate the "Ruff: Organize Imports" edit
        for kind in (
            CodeActionKind.SourceOrganizeImports,
            SOURCE_ORGANIZE_IMPORTS_RUFF,
            NOTEBOOK_SOURCE_ORGANIZE_IMPORTS,
            NOTEBOOK_SOURCE_ORGANIZE_IMPORTS_RUFF,
        ):
            if (
                params.context.only
                and len(params.context.only) == 1
                and kind in params.context.only
            ):
                workspace_edit = await _fix_document_impl(
                    document_from_kind(params.text_document.uri, kind),
                    settings,
                    only=["I001", "I002"],
                )
                if workspace_edit:
                    return [
                        CodeAction(
                            title="Ruff: Organize Imports",
                            kind=kind,
                            data=params.text_document.uri,
                            edit=workspace_edit,
                            diagnostics=[],
                        )
                    ]
                else:
                    return []

    # If the linter is enabled, generate the "Ruff: Fix All" edit.
    if lint_enable(settings) and settings["fixAll"]:
        for kind in (
```

```
            CodeActionKind.SourceFixAll,
            SOURCE_FIX_ALL_RUFF,
            NOTEBOOK_SOURCE_FIX_ALL,
            NOTEBOOK_SOURCE_FIX_ALL_RUFF,
        ):
            if (
                params.context.only
                and len(params.context.only) == 1
                and kind in params.context.only
            ):
                workspace_edit = await _fix_document_impl(
                    document_from_kind(params.text_document.uri, kind),
                    settings,
                )
                if workspace_edit:
                    return [
                        CodeAction(
                            title="Ruff: Fix All",
                            kind=kind,
                            data=params.text_document.uri,
                            edit=workspace_edit,
                            diagnostics=[
                                diagnostic
                                for diagnostic in params.context.diagnostics
                                if diagnostic.source == "Ruff"
                                and cast(DiagnosticData, diagnostic.data).get("fix")
                                is not None
                            ],
                        ),
                    ]
                else:
                    return []

    actions: list[CodeAction] = []

    # If the linter is enabled, add "Ruff: Autofix" for every fixable diagnostic.
    if lint_enable(settings) and settings.get("codeAction", {}).get(
        "fixViolation", {}
    ).get("enable", True):
        if not params.context.only or CodeActionKind.QuickFix in params.context.only:
            # This is a text document representing either a Python file or a
            # Notebook cell.
            text_document = LSP_SERVER.workspace.get_text_document(
                params.text_document.uri
            )
            for diagnostic in params.context.diagnostics:
                if diagnostic.source == "Ruff":
                    fix = cast(DiagnosticData, diagnostic.data).get("fix")
                    if fix is not None:
                        title: str
                        if fix.get("message"):
                            title = f"Ruff ({diagnostic.code}): {fix['message']}"
                        elif diagnostic.code:
                            title = f"Ruff: Fix {diagnostic.code}"
                        else:
                            title = "Ruff: Fix"

                        actions.append(
                            CodeAction(
                                title=title,
                                kind=CodeActionKind.QuickFix,
                                data=params.text_document.uri,
                                edit=_create_workspace_edit(
                                    text_document.uri, text_document.version, fix
```

```
                        ),
                        diagnostics=[diagnostic],
                    ),
                )

    # If the linter is enabled, add "Disable for this line" for every diagnostic.
    if lint_enable(settings) and settings.get("codeAction", {}).get(
        "disableRuleComment", {}
    ).get("enable", True):
        if not params.context.only or CodeActionKind.QuickFix in params.context.only:
            # This is a text document representing either a Python file or a
            # Notebook cell.
            text_document = LSP_SERVER.workspace.get_text_document(
                params.text_document.uri
            )
            lines: list[str] | None = None
            for diagnostic in params.context.diagnostics:
                if diagnostic.source == "Ruff":
                    noqa_row = cast(DiagnosticData, diagnostic.data).get("noqa_row")
                    if noqa_row is not None:
                        if lines is None:
                            lines = text_document.lines
                        line = lines[noqa_row - 1].rstrip("\r\n")

                        match = NOQA_REGEX.search(line)
                        if match and match.group("codes") is not None:
                            # `foo  # noqa: OLD` -> `foo  # noqa: OLD,NEW`
                            codes = match.group("codes") + f", {diagnostic.code}"
                            start, end = match.start("codes"), match.end("codes")
                            new_line = line[:start] + codes + line[end:]
                        elif match:
                            # `foo  # noqa` -> `foo  # noqa: NEW`
                            end = match.end("noqa")
                            new_line = line[:end] + f": {diagnostic.code}" + line[end:]
                        else:
                            # `foo` -> `foo  # noqa: NEW`
                            new_line = f"{line}  # noqa: {diagnostic.code}"
                        fix = Fix(
                            message=None,
                            applicability=None,
                            edits=[
                                Edit(
                                    content=new_line,
                                    location=Location(
                                        row=noqa_row,
                                        column=0,
                                    ),
                                    end_location=Location(
                                        row=noqa_row,
                                        column=len(line),
                                    ),
                                )
                            ],
                        )

                        title = f"Ruff ({diagnostic.code}): Disable for this line"

                        actions.append(
                            CodeAction(
                                title=title,
                                kind=CodeActionKind.QuickFix,
                                data=params.text_document.uri,
                                edit=_create_workspace_edit(
                                    text_document.uri, text_document.version, fix
```

```
                            ),
                            diagnostics=[diagnostic],
                        ),
                    )

    if settings["organizeImports"]:
        # Add "Ruff: Organize Imports" as a supported action.
        if not params.context.only or (
            CodeActionKind.SourceOrganizeImports in params.context.only
        ):
            if CLIENT_CAPABILITIES[CODE_ACTION_RESOLVE]:
                actions.append(
                    CodeAction(
                        title="Ruff: Organize Imports",
                        kind=CodeActionKind.SourceOrganizeImports,
                        data=params.text_document.uri,
                        edit=None,
                        diagnostics=[],
                    ),
                )
            else:
                workspace_edit = await _fix_document_impl(
                    Document.from_cell_or_text_uri(params.text_document.uri),
                    settings,
                    only=["I001", "I002"],
                )
                if workspace_edit:
                    actions.append(
                        CodeAction(
                            title="Ruff: Organize Imports",
                            kind=CodeActionKind.SourceOrganizeImports,
                            data=params.text_document.uri,
                            edit=workspace_edit,
                            diagnostics=[],
                        ),
                    )

    # If the linter is enabled, add "Ruff: Fix All" as a supported action.
    if lint_enable(settings) and settings["fixAll"]:
        if not params.context.only or (
            CodeActionKind.SourceFixAll in params.context.only
        ):
            if CLIENT_CAPABILITIES[CODE_ACTION_RESOLVE]:
                actions.append(
                    CodeAction(
                        title="Ruff: Fix All",
                        kind=CodeActionKind.SourceFixAll,
                        data=params.text_document.uri,
                        edit=None,
                        diagnostics=[],
                    ),
                )
            else:
                workspace_edit = await _fix_document_impl(
                    Document.from_cell_or_text_uri(params.text_document.uri),
                    settings,
                )
                if workspace_edit:
                    actions.append(
                        CodeAction(
                            title="Ruff: Fix All",
                            kind=CodeActionKind.SourceFixAll,
                            data=params.text_document.uri,
                            edit=workspace_edit,
```

```
                                diagnostics=[
                                    diagnostic
                                    for diagnostic in params.context.diagnostics
                                    if diagnostic.source == "Ruff"
                                    and cast(DiagnosticData, diagnostic.data).get("fix")
                                    is not None
                                ],
                            ),
                        )

    return actions if actions else None


@LSP_SERVER.feature(CODE_ACTION_RESOLVE)
async def resolve_code_action(params: CodeAction) -> CodeAction:
    """LSP handler for codeAction/resolve request."""
    # We set the `data` field to the document URI during codeAction request.
    document = Document.from_cell_or_text_uri(cast(str, params.data))

    settings = _get_settings_by_document(document.path)

    if (
        settings["organizeImports"]
        and params.kind == CodeActionKind.SourceOrganizeImports
    ):
        # Generate the "Organize Imports" edit
        params.edit = await _fix_document_impl(
            document, settings, only=["I001", "I002"]
        )

    elif (
        lint_enable(settings)
        and settings["fixAll"]
        and params.kind == CodeActionKind.SourceFixAll
    ):
        # Generate the "Fix All" edit.
        params.edit = await _fix_document_impl(document, settings)

    return params


@LSP_SERVER.command("ruff.applyAutofix")
async def apply_autofix(arguments: tuple[TextDocument]):
    uri = arguments[0]["uri"]
    document = Document.from_uri(uri)
    settings = _get_settings_by_document(document.path)
    if not lint_enable(settings):
        return None

    workspace_edit = await _fix_document_impl(document, settings)
    if workspace_edit is None:
        return None
    LSP_SERVER.apply_edit(workspace_edit, "Ruff: Fix all auto-fixable problems")


@LSP_SERVER.command("ruff.applyOrganizeImports")
async def apply_organize_imports(arguments: tuple[TextDocument]):
    uri = arguments[0]["uri"]
    document = Document.from_uri(uri)
    settings = _get_settings_by_document(document.path)
    workspace_edit = await _fix_document_impl(document, settings, only=
["I001", "I002"])
    if workspace_edit is None:
        return None
```

```
    LSP_SERVER.apply_edit(workspace_edit, "Ruff: Format imports")


@LSP_SERVER.command("ruff.applyFormat")
async def apply_format(arguments: tuple[TextDocument]):
    uri = arguments[0]["uri"]
    document = Document.from_uri(uri)
    settings = _get_settings_by_document(document.path)

    result = await _run_format_on_document(document, settings)
    if result is None:
        return None

    # For `ruff format`, 0 indicates successful completion, non-zero indicates
an error.
    if result.exit_code != 0:
        if result.stderr:
            show_error(f"Ruff: Format failed ({result.stderr.decode('utf-8')})")
        return None

    workspace_edit = _result_to_workspace_edit(document, result)
    if workspace_edit is None:
        return None
    LSP_SERVER.apply_edit(workspace_edit, "Ruff: Format document")


@LSP_SERVER.feature(TEXT_DOCUMENT_FORMATTING)
async def format_document(params: DocumentFormattingParams) -> list[TextEdit] | None:
    return await _format_document_impl(params, None)


@LSP_SERVER.feature(
    TEXT_DOCUMENT_RANGE_FORMATTING,
    DocumentRangeFormattingRegistrationOptions(
        document_selector=[
            TextDocumentFilter_Type1(language="python", scheme="file"),
            TextDocumentFilter_Type1(language="python", scheme="untitled"),
        ],
        ranges_support=False,
        work_done_progress=False,
    ),
)
async def format_document_range(
    params: DocumentRangeFormattingParams,
) -> list[TextEdit] | None:
    return await _format_document_impl(
        DocumentFormattingParams(
            params.text_document, params.options, params.work_done_token
        ),
        params.range,
    )


async def _format_document_impl(
    params: DocumentFormattingParams, range: Range | None
) -> list[TextEdit] | None:
    # For a Jupyter Notebook, this request can only format a single cell as the
    # request itself can only act on a text document. A cell in a Notebook is
    # represented as a text document. The "Notebook: Format notebook" action calls
    # this request for every cell.
    document = Document.from_cell_or_text_uri(params.text_document.uri)

    settings = _get_settings_by_document(document.path)
```

```
    # We don't support range formatting of notebooks yet but VS Code
    # doesn't seem to respect the document filter. For now, format the entire cell.
    range = None if document.kind is DocumentKind.Cell else range

    result = await _run_format_on_document(document, settings, range)
    if result is None:
        return None

    # For `ruff format`, 0 indicates successful completion, non-zero indicates
an error.
    if result.exit_code != 0:
        if result.stderr:
            show_error(f"Ruff: Format failed ({result.stderr.decode('utf-8')})")
        return None

    if not VERSION_REQUIREMENT_EMPTY_OUTPUT.contains(
        result.executable.version, prereleases=True
    ):
        if not result.stdout and document.source.strip():
            return None

    if document.kind is DocumentKind.Cell:
        return _result_single_cell_notebook_to_edits(document, result)
    else:
        return _fixed_source_to_edits(
            original_source=document.source, fixed_source=result.stdout.decode("utf-8")
        )


async def _fix_document_impl(
    document: Document,
    settings: WorkspaceSettings,
    *,
    only: Sequence[str] | None = None,
) -> WorkspaceEdit | None:
    result = await _run_check_on_document(
        document,
        settings,
        extra_args=["--fix"],
        only=only,
    )

    if result is None:
        return None

    # For `ruff check`, 0 indicates successful completion with no remaining
    # diagnostics, 1 indicates successful completion with remaining diagnostics,
    # and 2 indicates an error.
    if result.exit_code not in (0, 1):
        if result.stderr:
            show_error(f"Ruff: Fix failed ({result.stderr.decode('utf-8')})")
        return None

    return _result_to_workspace_edit(document, result)


def _result_to_workspace_edit(
    document: Document, result: ExecutableResult | None
) -> WorkspaceEdit | None:
    """Converts a run result to a WorkspaceEdit."""
    if result is None:
        return None

    if not VERSION_REQUIREMENT_EMPTY_OUTPUT.contains(
```

```
        result.executable.version, prereleases=True
    ):
        if not result.stdout and document.source.strip():
            return None

    if document.kind is DocumentKind.Text:
        edits = _fixed_source_to_edits(
            original_source=document.source, fixed_source=result.stdout.decode("utf-8")
        )
        return WorkspaceEdit(
            document_changes=[
                _create_text_document_edit(document.uri, document.version, edits)
            ]
        )
    elif document.kind is DocumentKind.Notebook:
        notebook_document = LSP_SERVER.workspace.get_notebook_document(
            notebook_uri=document.uri
        )
        if notebook_document is None:
            log_warning(f"No notebook document found for {document.uri!r}")
            return None

        output_notebook_cells = cast(
            Notebook, json.loads(result.stdout.decode("utf-8"))
        )["cells"]
        if len(output_notebook_cells) != len(notebook_document.cells):
            log_warning(
                f"Number of cells in the output notebook doesn't match the number of "
                f"cells in the input notebook. Input: {len(notebook_document.cells)}, "
                f"Output: {len(output_notebook_cells)}"
            )
            return None

        cell_document_changes: list[TextDocumentEdit] = []
        for cell_idx, cell in enumerate(notebook_document.cells):
            if cell.kind is not NotebookCellKind.Code:
                continue
            cell_document = LSP_SERVER.workspace.get_text_document(cell.document)
            edits = _fixed_source_to_edits(
                original_source=cell_document.source,
                fixed_source=output_notebook_cells[cell_idx]["source"],
            )
            cell_document_changes.append(
                _create_text_document_edit(
                    cell_document.uri,
                    cell_document.version,
                    edits,
                )
            )

        return WorkspaceEdit(document_changes=list(cell_document_changes))
    elif document.kind is DocumentKind.Cell:
        text_edits = _result_single_cell_notebook_to_edits(document, result)
        if text_edits is None:
            return None
        return WorkspaceEdit(
            document_changes=[
                _create_text_document_edit(document.uri, document.version, text_edits)
            ]
        )
    else:
        assert_never(document.kind)
```

```
def _result_single_cell_notebook_to_edits(
    document: Document, result: ExecutableResult
) -> list[TextEdit] | None:
    """Converts a run result to a list of TextEdits.

    The result is expected to be a single cell Notebook Document.
    """
    output_notebook = cast(Notebook, json.loads(result.stdout.decode("utf-8")))
    # The input notebook contained only one cell, so the output notebook should
    # also contain only one cell.
    output_cell = next(iter(output_notebook["cells"]), None)
    if output_cell is None or output_cell["cell_type"] != "code":
        log_warning(
            f"Unexpected output working with a notebook cell: {output_notebook}"
        )
        return None
    # We can't use the `document.source` here because it's in the Notebook format
    # i.e., it's a JSON string containing a single cell with the source.
    original_source = LSP_SERVER.workspace.get_text_document(document.uri).source
    return _fixed_source_to_edits(
        original_source=original_source, fixed_source=output_cell["source"]
    )


def _fixed_source_to_edits(
    *, original_source: str, fixed_source: str | list[str]
) -> list[TextEdit]:
    """Converts the fixed source to a list of TextEdits.

    If the fixed source is a list of strings, it is joined together to form a single
    string with an assumption that the line endings are part of the strings itself.
    """
    if isinstance(fixed_source, list):
        fixed_source = "".join(fixed_source)

    new_source = _match_line_endings(original_source, fixed_source)

    if new_source == original_source:
        return []

    # Reduce the text edit by omitting the common suffix and postfix (lines)
    # from the text edit. I chose this basic diffing because "proper" diffing has
    # the downside that it can be very slow in some cases. Black uses a
diffing approach
    # that takes time into consideration, but it requires spawning a thread to stop
    # the diffing after a given time, which feels very heavy weight.
    # This basic "diffing" has a guaranteed `O(n)` runtime and is sufficient to
    # prevent transmitting the entire source document when formatting a range
    # or formatting a document where most of the code remains unchanged.
    #
    # https://github.com/microsoft/vscode-black-
formatter/blob/main/bundled/tool/lsp_edit_utils.py
    new_lines = new_source.splitlines(True)
    original_lines = original_source.splitlines(True)

    start_offset = 0
    end_offset = 0

    for new_line, original_line in zip(new_lines, original_lines):
        if new_line == original_line:
            start_offset += 1
        else:
            break
```

```python
    for new_line, original_line in zip(
        reversed(new_lines[start_offset:]), reversed(original_lines[start_offset:])
    ):
        if new_line == original_line:
            end_offset += 1
        else:
            break

    trimmed_new_source = "".join(new_lines[start_offset : len(new_lines) - end_offset])

    return [
        TextEdit(
            range=Range(
                start=Position(line=start_offset, character=0),
                end=Position(line=len(original_lines) - end_offset, character=0),
            ),
            new_text=trimmed_new_source,
        )
    ]


def _create_text_document_edit(
    uri: str, version: int | None, edits: Sequence[TextEdit | AnnotatedTextEdit]
) -> TextDocumentEdit:
    return TextDocumentEdit(
        text_document=OptionalVersionedTextDocumentIdentifier(
            uri=uri,
            version=0 if version is None else version,
        ),
        edits=list(edits),
    )


def _create_workspace_edit(uri: str, version: int | None, fix: Fix) -> WorkspaceEdit:
    return WorkspaceEdit(
        document_changes=[
            TextDocumentEdit(
                text_document=OptionalVersionedTextDocumentIdentifier(
                    uri=uri,
                    version=0 if version is None else version,
                ),
                edits=[
                    TextEdit(
                        range=Range(
                            start=Position(
                                line=edit["location"]["row"] - 1,
                                character=edit["location"]["column"],
                            ),
                            end=Position(
                                line=edit["end_location"]["row"] - 1,
                                character=edit["end_location"]["column"],
                            ),
                        ),
                        new_text=edit["content"],
                    )
                    for edit in fix["edits"]
                ],
            )
        ],
    )


def _get_line_endings(text: str) -> str | None:
    """Returns line endings used in the text."""
```

```
    for i in range(len(text)):
        if text[i] == "\r":
            if i < len(text) - 1 and text[i + 1] == "\n":
                return "\r\n"  # CLRF
            else:
                return "\r"  # CR
        elif text[i] == "\n":
            return "\n"  # LF
    return None  # No line ending found


def _match_line_endings(original_source: str, fixed_source: str) -> str:
    """Ensures that the edited text line endings matches the document line endings."""
    expected = _get_line_endings(original_source)
    actual = _get_line_endings(fixed_source)
    if actual is None or expected is None or actual == expected:
        return fixed_source
    return fixed_source.replace(actual, expected)


async def run_path(
    program: str,
    argv: Sequence[str],
    *,
    source: str,
    cwd: str | None = None,
) -> RunResult:
    """Runs as an executable."""
    log_to_output(f"Running Ruff with: {program} {argv}")

    process = await asyncio.create_subprocess_exec(
        program,
        *argv,
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.PIPE,
        stdin=asyncio.subprocess.PIPE,
        cwd=cwd,
    )
    result = RunResult(
        *await process.communicate(input=source.encode("utf-8")),
        exit_code=await process.wait(),
    )

    if result.stderr:
        log_to_output(result.stderr.decode("utf-8"))

    return result


###
# Lifecycle.
###


@LSP_SERVER.feature(INITIALIZE)
def initialize(params: InitializeParams) -> None:
    """LSP handler for initialize request."""
    # Extract client capabilities.
    CLIENT_CAPABILITIES[CODE_ACTION_RESOLVE] = _supports_code_action_resolve(
        params.capabilities
    )

    # Extract `settings` from the initialization options.
    workspace_settings: list[WorkspaceSettings] | WorkspaceSettings | None = (
```

```
        params.initialization_options or {}
    ).get(
        "settings",
    )
    global_settings: UserSettings | None = (params.initialization_options or {}).get(
        "globalSettings", {}
    )

    log_to_output(
        f"Workspace settings: "
        f"{json.dumps(workspace_settings, indent=4, ensure_ascii=False)}"
    )
    log_to_output(
        f"Global settings: "
        f"{json.dumps(global_settings, indent=4, ensure_ascii=False)}"
    )

    # Preserve any "global" settings.
    if global_settings:
        GLOBAL_SETTINGS.update(global_settings)
    elif isinstance(workspace_settings, dict):
        # In Sublime Text, Neovim, and probably others, we're passed a single
        # `settings`, which we'll treat as defaults for any future files.
        GLOBAL_SETTINGS.update(workspace_settings)

    # Update workspace settings.
    settings: list[WorkspaceSettings]
    if isinstance(workspace_settings, dict):
        settings = [workspace_settings]
    elif isinstance(workspace_settings, list):
        # In VS Code, we're passed a list of `settings`, one for each workspace folder.
        settings = workspace_settings
    else:
        settings = []

    _update_workspace_settings(settings)


def _supports_code_action_resolve(capabilities: ClientCapabilities) -> bool:
    """Returns True if the client supports codeAction/resolve request for edits."""
    if capabilities.text_document is None:
        return False

    if capabilities.text_document.code_action is None:
        return False

    if capabilities.text_document.code_action.resolve_support is None:
        return False

    return "edit" in capabilities.text_document.code_action.resolve_support.properties


###
# Settings.
###


def _get_global_defaults() -> UserSettings:
    settings: UserSettings = {
        "codeAction": GLOBAL_SETTINGS.get("codeAction", {}),
        "fixAll": GLOBAL_SETTINGS.get("fixAll", True),
        "format": GLOBAL_SETTINGS.get("format", {}),
        "ignoreStandardLibrary": GLOBAL_SETTINGS.get("ignoreStandardLibrary", True),
        "importStrategy": GLOBAL_SETTINGS.get("importStrategy", "fromEnvironment"),
```

```python
        "interpreter": GLOBAL_SETTINGS.get("interpreter", [sys.executable]),
        "lint": GLOBAL_SETTINGS.get("lint", {}),
        "logLevel": GLOBAL_SETTINGS.get("logLevel", "error"),
        "organizeImports": GLOBAL_SETTINGS.get("organizeImports", True),
        "path": GLOBAL_SETTINGS.get("path", []),
    }

    # Deprecated: use `lint.args` instead.
    if "args" in GLOBAL_SETTINGS:
        settings["args"] = GLOBAL_SETTINGS["args"]

    # Deprecated: use `lint.run` instead.
    if "run" in GLOBAL_SETTINGS:
        settings["run"] = GLOBAL_SETTINGS["run"]

    return settings


def _update_workspace_settings(settings: list[WorkspaceSettings]) -> None:
    if not settings:
        workspace_path = os.getcwd()
        WORKSPACE_SETTINGS[workspace_path] = {
            **_get_global_defaults(),  # type: ignore[misc]
            "cwd": workspace_path,
            "workspacePath": workspace_path,
            "workspace": uris.from_fs_path(workspace_path),
        }
        return None

    for setting in settings:
        if "workspace" in setting:
            workspace_path = uris.to_fs_path(setting["workspace"])
            WORKSPACE_SETTINGS[workspace_path] = {
                **_get_global_defaults(),  # type: ignore[misc]
                **setting,
                "cwd": workspace_path,
                "workspacePath": workspace_path,
                "workspace": setting["workspace"],
            }
        else:
            workspace_path = os.getcwd()
            WORKSPACE_SETTINGS[workspace_path] = {
                **_get_global_defaults(),  # type: ignore[misc]
                **setting,
                "cwd": workspace_path,
                "workspacePath": workspace_path,
                "workspace": uris.from_fs_path(workspace_path),
            }


def _get_document_key(document_path: str) -> str | None:
    document_workspace = Path(document_path)
    workspaces = {s["workspacePath"] for s in WORKSPACE_SETTINGS.values()}

    while document_workspace != document_workspace.parent:
        if str(document_workspace) in workspaces:
            return str(document_workspace)
        document_workspace = document_workspace.parent
    return None


def _get_settings_by_document(document_path: str) -> WorkspaceSettings:
    key = _get_document_key(document_path)
    if key is None:
```

```python
        # This is either a non-workspace file or there is no workspace.
        workspace_path = os.fspath(Path(document_path).parent)
        return {
            **_get_global_defaults(),  # type: ignore[misc]
            "cwd": None,
            "workspacePath": workspace_path,
            "workspace": uris.from_fs_path(workspace_path),
        }

    return WORKSPACE_SETTINGS[str(key)]


###
# Internal execution APIs.
###


class Executable(NamedTuple):
    path: str
    """The path to the executable."""

    version: Version
    """The version of the executable."""


class ExecutableResult(NamedTuple):
    executable: Executable
    """The executable."""

    stdout: bytes
    """The stdout of running the executable."""

    stderr: bytes
    """The stderr of running the executable."""

    exit_code: int
    """The exit code of running the executable."""


def _find_ruff_binary(
    settings: WorkspaceSettings, version_requirement: SpecifierSet | None
) -> Executable:
    """Returns the executable along with its version.

    If the executable doesn't meet the version requirement, raises a RuntimeError and
    displays an error message.
    """
    path = _find_ruff_binary_path(settings)

    version = _executable_version(path)
    if version_requirement and not version_requirement.contains(
        version, prereleases=True
    ):
        message = f"Ruff {version_requirement} required, but found {version} at {path}"
        show_error(message)
        raise RuntimeError(message)
    log_to_output(f"Found ruff {version} at {path}")

    return Executable(path, version)


def _find_ruff_binary_path(settings: WorkspaceSettings) -> str:
    """Returns the path to the executable."""
    bundle = get_bundle()
```

```python
    if settings["path"]:
        # 'path' setting takes priority over everything.
        paths = settings["path"]
        if isinstance(paths, str):
            paths = [paths]
        for path in paths:
            path = os.path.expanduser(os.path.expandvars(path))
            if os.path.exists(path):
                log_to_output(f"Using 'path' setting: {path}")
                return path
        else:
            log_to_output(f"Could not find executable in 'path': {settings['path']}")

    if settings["importStrategy"] == "useBundled" and bundle:
        # If we're loading from the bundle, use the absolute path.
        log_to_output(f"Using bundled executable: {bundle}")
        return bundle

    if settings["interpreter"] and not utils.is_current_interpreter(
        settings["interpreter"][0]
    ):
        # If there is a different interpreter set, find its script path.
        if settings["interpreter"][0] not in INTERPRETER_PATHS:
            INTERPRETER_PATHS[settings["interpreter"][0]] = utils.scripts(
                os.path.expanduser(os.path.expandvars(settings["interpreter"][0]))
            )

        path = os.path.join(INTERPRETER_PATHS[settings["interpreter"][0]], TOOL_MODULE)
    else:
        path = os.path.join(sysconfig.get_path("scripts"), TOOL_MODULE)

    # First choice: the executable in the current interpreter's scripts directory.
    if os.path.exists(path):
        log_to_output(f"Using interpreter executable: {path}")
        return path
    else:
        log_to_output(f"Interpreter executable ({path}) not found")

    # Second choice: the executable in the global environment.
    environment_path = shutil.which("ruff")
    if environment_path:
        log_to_output(f"Using environment executable: {environment_path}")
        return environment_path

    # Third choice: bundled executable.
    if bundle:
        log_to_output(f"Falling back to bundled executable: {bundle}")
        return bundle

    # Last choice: just return the expected path for the current interpreter.
    log_to_output(f"Unable to find interpreter executable: {path}")
    return path


def _executable_version(executable: str) -> Version:
    """Returns the version of the executable."""
    # If the user change the file (e.g. `pip install -U ruff`), invalidate the cache
    modified = Path(executable).stat().st_mtime
    if (
        executable not in EXECUTABLE_VERSIONS
        or EXECUTABLE_VERSIONS[executable].modified != modified
    ):
        version = utils.version(executable)
```

```
        log_to_output(f"Inferred version {version} for: {executable}")
        EXECUTABLE_VERSIONS[executable] = VersionModified(version, modified)
    return EXECUTABLE_VERSIONS[executable].version


async def _run_check_on_document(
    document: Document,
    settings: WorkspaceSettings,
    *,
    extra_args: Sequence[str] = [],
    only: Sequence[str] | None = None,
) -> ExecutableResult | None:
    """Runs the Ruff `check` subcommand  on the given document source."""
    if settings.get("ignoreStandardLibrary", True) and document.is_stdlib_file():
        log_warning(f"Skipping standard library file: {document.path}")
        return None

    executable = _find_ruff_binary(settings, VERSION_REQUIREMENT_LINTER)
    argv: list[str] = CHECK_ARGS + list(extra_args)

    skip_next_arg = False
    for arg in lint_args(settings):
        if skip_next_arg:
            skip_next_arg = False
            continue
        if arg in UNSUPPORTED_CHECK_ARGS:
            log_to_output(f"Ignoring unsupported argument: {arg}")
            continue
        # If we're trying to run a single rule, we need to make sure to skip any of the
        # arguments that would override it.
        if only is not None:
            # Case 1: Option and its argument as separate items
            # (e.g. `["--select", "F821"]`).
            if arg in ("--select", "--extend-select", "--ignore", "--extend-ignore"):
                # Skip the following argument assuming it's a list of rules.
                skip_next_arg = True
                continue
            # Case 2: Option and its argument as a single item
            # (e.g. `["--select=F821"]`).
            elif arg.startswith(
                ("--select=", "--extend-select=", "--ignore=", "--extend-ignore=")
            ):
                continue
        argv.append(arg)

    # If the Ruff version is not sufficiently recent, use the deprecated `--format`
    # argument instead of `--output-format`.
    if not VERSION_REQUIREMENT_OUTPUT_FORMAT.contains(
        executable.version, prereleases=True
    ):
        index = argv.index("--output-format")
        argv.pop(index)
        argv.insert(index, "--format")

    # If we're trying to run a single rule, add it to the command line.
    if only is not None:
        for rule in only:
            argv += ["--select", rule]

    # Provide the document filename.
    argv += ["--stdin-filename", document.path]

    return ExecutableResult(
        executable,
```

```
        *await run_path(
            executable.path,
            argv,
            cwd=settings["cwd"],
            source=document.source,
        ),
    )


async def _run_format_on_document(
    document: Document, settings: WorkspaceSettings, format_range: Range | None = None
) -> ExecutableResult | None:
    """Runs the Ruff `format` subcommand on the given document source."""
    if settings.get("ignoreStandardLibrary", True) and document.is_stdlib_file():
        log_warning(f"Skipping standard library file: {document.path}")
        return None

    version_requirement = (
        VERSION_REQUIREMENT_FORMATTER
        if format_range is None
        else VERSION_REQUIREMENT_RANGE_FORMATTING
    )
    executable = _find_ruff_binary(settings, version_requirement)
    argv: list[str] = [
        "format",
        "--force-exclude",
        "--quiet",
        "--stdin-filename",
        document.path,
    ]

    if format_range:
        codec = PositionCodec(PositionEncodingKind.Utf16)
        format_range = codec.range_from_client_units(
            document.source.splitlines(True), format_range
        )

        argv.extend(
            [
                "--range",
                f"{format_range.start.line + 1}:{format_range.start.character + 1}-
{format_range.end.line + 1}:{format_range.end.character + 1}",  # noqa: E501
            ]
        )

    for arg in settings.get("format", {}).get("args", []):
        if arg in UNSUPPORTED_FORMAT_ARGS:
            log_to_output(f"Ignoring unsupported argument: {arg}")
        else:
            argv.append(arg)

    return ExecutableResult(
        executable,
        *await run_path(
            executable.path,
            argv,
            cwd=settings["cwd"],
            source=document.source,
        ),
    )


async def _run_subcommand_on_document(
    document: workspace.TextDocument,
```

```python
    version_requirement: SpecifierSet,
    *,
    args: Sequence[str],
) -> ExecutableResult:
    """Runs the tool subcommand on the given document."""
    settings = _get_settings_by_document(document.path)

    executable = _find_ruff_binary(settings, version_requirement)
    argv: list[str] = list(args)

    return ExecutableResult(
        executable,
        *await run_path(
            executable.path,
            argv,
            cwd=settings["cwd"],
            source=document.source,
        ),
    )


###
# Logging.
###


def log_to_output(message: str) -> None:
    LSP_SERVER.show_message_log(message, MessageType.Log)


def show_error(message: str) -> None:
    """Show a pop-up with an error. Only use for critical errors."""
    LSP_SERVER.show_message_log(message, MessageType.Error)
    LSP_SERVER.show_message(message, MessageType.Error)


def log_warning(message: str) -> None:
    LSP_SERVER.show_message_log(message, MessageType.Warning)
    if os.getenv("LS_SHOW_NOTIFICATION", "off") in ["onWarning", "always"]:
        LSP_SERVER.show_message(message, MessageType.Warning)


def log_always(message: str) -> None:
    LSP_SERVER.show_message_log(message, MessageType.Info)
    if os.getenv("LS_SHOW_NOTIFICATION", "off") in ["always"]:
        LSP_SERVER.show_message(message, MessageType.Info)


###
# Bundled mode.
###

_BUNDLED_PATH: str | None = None


def set_bundle(path: str) -> None:
    """Sets the path to the bundled Ruff executable."""
    global _BUNDLED_PATH
    _BUNDLED_PATH = path


def get_bundle() -> str | None:
    """Returns the path to the bundled Ruff executable."""
    return _BUNDLED_PATH
```

```
###
# Start up.
###


def start() -> None:
    LSP_SERVER.start_io()


if __name__ == "__main__":
    start()
}
```

ruff_lsp/settings.py

```
from __future__ import annotations

import enum

from typing_extensions import Literal, TypedDict


@enum.unique
class Run(str, enum.Enum):
    """When to run Ruff."""

    OnType = "onType"
    """Run Ruff on every keystroke."""

    OnSave = "onSave"
    """Run Ruff on save."""


class UserSettings(TypedDict, total=False):
    """Settings for the Ruff Language Server."""

    logLevel: Literal["error", "warning", "info", "debug"]
    """The log level for the Ruff server. Defaults to "error"."""

    path: list[str]
    """Path to a custom `ruff` executable."""

    interpreter: list[str]
    """Path to a Python interpreter to use to run the linter server."""

    importStrategy: Literal["fromEnvironment", "useBundled"]
    """Strategy for loading the `ruff` executable."""

    codeAction: CodeActions
    """Settings for the `source.codeAction` capability."""

    organizeImports: bool
    """Whether to register Ruff as capable of handling `source.organizeImports`."""

    fixAll: bool
    """Whether to register Ruff as capable of handling `source.fixAll`."""

    lint: Lint
```

```python
    """Settings specific to lint capabilities."""

    format: Format
    """Settings specific to format capabilities."""

    ignoreStandardLibrary: bool
    """Whether to ignore files that are inferred to be part of the standard library."""

    showSyntaxErrors: bool
    """Whether to show syntax error diagnostics."""

    # Deprecated: use `lint.args` instead.
    args: list[str]
    """Additional command-line arguments to pass to `ruff check`."""

    # Deprecated: use `lint.run` instead.
    run: Run
    """Run Ruff on every keystroke (`onType`) or on save (`onSave`)."""


class WorkspaceSettings(TypedDict, UserSettings):
    cwd: str | None
    """The current working directory for the workspace."""

    workspacePath: str
    """The path to the workspace."""

    workspace: str
    """The workspace name."""


class CodeActions(TypedDict, total=False):
    fixViolation: CodeAction
    """Code action to fix a violation."""

    disableRuleComment: CodeAction
    """Code action to disable the rule on the current line."""


class CodeAction(TypedDict, total=False):
    enable: bool
    """Whether to enable the code action."""


class Lint(TypedDict, total=False):
    enable: bool
    """Whether to enable linting."""

    args: list[str]
    """Additional command-line arguments to pass to `ruff check`."""

    run: Run
    """Run Ruff on every keystroke (`onType`) or on save (`onSave`)."""


class Format(TypedDict, total=False):
    args: list[str]
    """Additional command-line arguments to pass to `ruff format`."""


def lint_args(settings: UserSettings) -> list[str]:
    """Get the `lint.args` setting from the user settings."""
    if "lint" in settings and "args" in settings["lint"]:
        return settings["lint"]["args"]
```

```
    elif "args" in settings:
        return settings["args"]
    else:
        return []


def lint_run(settings: UserSettings) -> Run:
    """Get the `lint.run` setting from the user settings."""
    if "lint" in settings and "run" in settings["lint"]:
        return Run(settings["lint"]["run"])
    elif "run" in settings:
        return Run(settings["run"])
    else:
        return Run.OnType


def lint_enable(settings: UserSettings) -> bool:
    """Get the `lint.enable` setting from the user settings."""
    if "lint" in settings and "enable" in settings["lint"]:
        return settings["lint"]["enable"]
    else:
        return True
}
```

ruff_lsp/utils.py

```
"""Utility functions and classes for use with running tools over LSP."""

from __future__ import annotations

import os
import os.path
import pathlib
import site
import subprocess
import sys
import sysconfig
from typing import Any, NamedTuple

from packaging.version import Version


def as_list(content: Any | list[Any] | tuple[Any, ...]) -> list[Any]:
    """Ensures we always get a list"""
    if isinstance(content, (list, tuple)):
        return list(content)
    return [content]


def _get_sys_config_paths() -> list[str]:
    """Returns paths from sysconfig.get_paths()."""
    return [
        path
        for group, path in sysconfig.get_paths().items()
        if group not in ["data", "platdata", "scripts"]
    ]


def _get_extensions_dir() -> list[str]:
    """This is the extensions folder under ~/.vscode or ~/.vscode-server."""
```

```
    # The path here is calculated relative to the tool
    # this is because users can launch VS Code with custom
    # extensions folder using the --extensions-dir argument
    path = pathlib.Path(__file__).parent.parent.parent.parent
    #                              ^     bundled  ^  extensions
    #                            tool        <extension>
    if path.name == "extensions":
        return [os.fspath(path)]
    return []


_stdlib_paths = set(
    str(pathlib.Path(p).resolve())
    for p in (
        as_list(site.getsitepackages())
        + as_list(site.getusersitepackages())
        + _get_sys_config_paths()
        + _get_extensions_dir()
    )
)


def is_same_path(file_path1: str, file_path2: str) -> bool:
    """Returns true if two paths are the same."""
    return pathlib.Path(file_path1) == pathlib.Path(file_path2)


def normalize_path(file_path: str) -> str:
    """Returns normalized path."""
    return str(pathlib.Path(file_path).resolve())


def is_current_interpreter(executable: str) -> bool:
    """Returns true if the executable path is same as the current interpreter."""
    return is_same_path(executable, sys.executable)


def is_stdlib_file(file_path: str) -> bool:
    """Return True if the file belongs to the standard library."""
    normalized_path = str(pathlib.Path(file_path).resolve())
    return any(normalized_path.startswith(path) for path in _stdlib_paths)


def scripts(interpreter: str) -> str:
    """Returns the absolute path to an interpreter's scripts directory."""
    return (
        subprocess.check_output(
            [
                interpreter,
                "-c",
                "import sysconfig; print(sysconfig.get_path('scripts'))",
            ]
        )
        .decode()
        .strip()
    )


def version(executable: str) -> Version:
    """Returns the version of the executable at the given path."""
    output = subprocess.check_output([executable, "--version"]).decode().strip()
    version = output.replace("ruff ", "")  # no removeprefix in 3.7 :/
    return Version(version)
```

```
class RunResult(NamedTuple):
    """Object to hold result from running tool."""

    stdout: bytes
    """The stdout of running the executable."""

    stderr: bytes
    """The stderr of running the executable."""

    exit_code: int
    """The exit code of running the executable."""
}
```

Chapter 2.0.0

tests

Chapter 2.1.0

tests/client

tests/client/constants.py

```
"""Constants for use with tests."""

from __future__ import annotations

import pathlib

TEST_ROOT = pathlib.Path(__file__).parent.parent
PROJECT_ROOT = TEST_ROOT.parent
}
```

tests/client/defaults.py

```
"""Default initialize request params."""

from __future__ import annotations

import os
from typing import Any, Mapping
```

```
from tests.client.constants import PROJECT_ROOT
from tests.client.utils import as_uri

VSCODE_DEFAULT_INITIALIZE: Mapping[str, Any] = {
    "processId": os.getpid(),
    "clientInfo": {"name": "vscode", "version": "1.45.0"},
    "rootPath": str(PROJECT_ROOT),
    "rootUri": as_uri(str(PROJECT_ROOT)),
    "capabilities": {
        "workspace": {
            "applyEdit": True,
            "workspaceEdit": {
                "documentChanges": True,
                "resourceOperations": ["create", "rename", "delete"],
                "failureHandling": "textOnlyTransactional",
            },
            "didChangeConfiguration": {"dynamicRegistration": True},
            "didChangeWatchedFiles": {"dynamicRegistration": True},
            "symbol": {
                "dynamicRegistration": True,
                "symbolKind": {
                    "valueSet": [
                        1,
                        2,
                        3,
                        4,
                        5,
                        6,
                        7,
                        8,
                        9,
                        10,
                        11,
                        12,
                        13,
                        14,
                        15,
                        16,
                        17,
                        18,
                        19,
                        20,
                        21,
                        22,
                        23,
                        24,
                        25,
                        26,
                    ]
                },
                "tagSupport": {"valueSet": [1]},
            },
            "executeCommand": {"dynamicRegistration": True},
            "configuration": True,
            "workspaceFolders": True,
        },
        "textDocument": {
            "publishDiagnostics": {
                "relatedInformation": True,
                "versionSupport": False,
                "tagSupport": {"valueSet": [1, 2]},
                "complexDiagnosticCodeSupport": True,
            },
            "synchronization": {
```

```
                "dynamicRegistration": True,
                "willSave": True,
                "willSaveWaitUntil": True,
                "didSave": True,
            },
            "completion": {
                "dynamicRegistration": True,
                "contextSupport": True,
                "completionItem": {
                    "snippetSupport": True,
                    "commitCharactersSupport": True,
                    "documentationFormat": ["markdown", "plaintext"],
                    "deprecatedSupport": True,
                    "preselectSupport": True,
                    "tagSupport": {"valueSet": [1]},
                    "insertReplaceSupport": True,
                },
                "completionItemKind": {
                    "valueSet": [
                        1,
                        2,
                        3,
                        4,
                        5,
                        6,
                        7,
                        8,
                        9,
                        10,
                        11,
                        12,
                        13,
                        14,
                        15,
                        16,
                        17,
                        18,
                        19,
                        20,
                        21,
                        22,
                        23,
                        24,
                        25,
                    ]
                },
            },
            "hover": {
                "dynamicRegistration": True,
                "contentFormat": ["markdown", "plaintext"],
            },
            "signatureHelp": {
                "dynamicRegistration": True,
                "signatureInformation": {
                    "documentationFormat": ["markdown", "plaintext"],
                    "parameterInformation": {"labelOffsetSupport": True},
                },
                "contextSupport": True,
            },
            "definition": {"dynamicRegistration": True, "linkSupport": True},
            "references": {"dynamicRegistration": True},
            "documentHighlight": {"dynamicRegistration": True},
            "documentSymbol": {
                "dynamicRegistration": True,
```

```
                "symbolKind": {
                    "valueSet": [
                        1,
                        2,
                        3,
                        4,
                        5,
                        6,
                        7,
                        8,
                        9,
                        10,
                        11,
                        12,
                        13,
                        14,
                        15,
                        16,
                        17,
                        18,
                        19,
                        20,
                        21,
                        22,
                        23,
                        24,
                        25,
                        26,
                    ]
                },
                "hierarchicalDocumentSymbolSupport": True,
                "tagSupport": {"valueSet": [1]},
            },
            "codeAction": {
                "dynamicRegistration": True,
                "isPreferredSupport": True,
                "codeActionLiteralSupport": {
                    "codeActionKind": {
                        "valueSet": [
                            "",
                            "quickfix",
                            "refactor",
                            "refactor.extract",
                            "refactor.inline",
                            "refactor.rewrite",
                            "source",
                            "source.organizeImports",
                        ]
                    }
                },
            },
            "codeLens": {"dynamicRegistration": True},
            "formatting": {"dynamicRegistration": True},
            "rangeFormatting": {"dynamicRegistration": True},
            "onTypeFormatting": {"dynamicRegistration": True},
            "rename": {"dynamicRegistration": True, "prepareSupport": True},
            "documentLink": {
                "dynamicRegistration": True,
                "tooltipSupport": True,
            },
            "typeDefinition": {
                "dynamicRegistration": True,
                "linkSupport": True,
            },
```

```
            "implementation": {
                "dynamicRegistration": True,
                "linkSupport": True,
            },
            "colorProvider": {"dynamicRegistration": True},
            "foldingRange": {
                "dynamicRegistration": True,
                "rangeLimit": 5000,
                "lineFoldingOnly": True,
            },
            "declaration": {"dynamicRegistration": True, "linkSupport": True},
            "selectionRange": {"dynamicRegistration": True},
        },
        "window": {"workDoneProgress": True},
    },
    "trace": "verbose",
    "workspaceFolders": [{"uri": as_uri(str(PROJECT_ROOT)), "name": "my_project"}],
    "initializationOptions": {},
}
}
```

tests/client/session.py

```
"""LSP session client for testing."""

from __future__ import annotations

import os
import subprocess
import sys
from concurrent.futures import Future, ThreadPoolExecutor
from threading import Event
from typing import Callable

from pylsp_jsonrpc.dispatchers import MethodDispatcher
from pylsp_jsonrpc.endpoint import Endpoint
from pylsp_jsonrpc.streams import JsonRpcStreamReader, JsonRpcStreamWriter

from tests.client.defaults import VSCODE_DEFAULT_INITIALIZE
from tests.client.utils import unwrap

LSP_EXIT_TIMEOUT = 5000


PUBLISH_DIAGNOSTICS = "textDocument/publishDiagnostics"
WINDOW_LOG_MESSAGE = "window/logMessage"
WINDOW_SHOW_MESSAGE = "window/showMessage"


class LspSession(MethodDispatcher):
    """Send and Receive messages over LSP."""

    def __init__(self, cwd: str, module: str):
        self.cwd = cwd
        self.module = module

        self._endpoint: Endpoint
        self._thread_pool: ThreadPoolExecutor = ThreadPoolExecutor()
        self._sub: subprocess.Popen | None = None
        self._reader: JsonRpcStreamReader | None = None
```

```
        self._writer: JsonRpcStreamWriter | None = None
        self._notification_callbacks: dict[str, Callable] = {}

    def __enter__(self):
        """Context manager entrypoint.

        shell=True needed for pytest-cov to work in subprocess.
        """
        self._sub = subprocess.Popen(
            [sys.executable, "-m", str(self.module)],
            stdout=subprocess.PIPE,
            stdin=subprocess.PIPE,
            bufsize=0,
            cwd=self.cwd,
            env=os.environ,
        )

        self._writer = JsonRpcStreamWriter(self._sub.stdin)
        self._reader = JsonRpcStreamReader(self._sub.stdout)

        dispatcher = {
            PUBLISH_DIAGNOSTICS: self._publish_diagnostics,
            WINDOW_SHOW_MESSAGE: self._window_show_message,
            WINDOW_LOG_MESSAGE: self._window_log_message,
        }
        self._endpoint = Endpoint(dispatcher, self._writer.write)
        self._thread_pool.submit(self._reader.listen, self._endpoint.consume)
        return self

    def __exit__(self, typ, value, _tb):
        self.shutdown(True)
        unwrap(self._sub).terminate()
        unwrap(self._sub).wait()
        self._endpoint.shutdown()  # type: ignore[union-attr]
        self._thread_pool.shutdown()
        unwrap(self._writer).close()  # type: ignore[attr-defined]
        unwrap(self._reader).close()  # type: ignore[attr-defined]

    def initialize(
        self,
        initialize_params=None,
        process_server_capabilities=None,
    ):
        """Sends the initialize request to LSP server."""
        if initialize_params is None:
            initialize_params = VSCODE_DEFAULT_INITIALIZE
        server_initialized = Event()

        def _after_initialize(fut):
            if process_server_capabilities:
                process_server_capabilities(fut.result())
            self.initialized()
            server_initialized.set()

        self._send_request(
            "initialize",
            params=(
                initialize_params
                if initialize_params is not None
                else VSCODE_DEFAULT_INITIALIZE
            ),
            handle_response=_after_initialize,
        )
```

```
        server_initialized.wait()

    def initialized(self, initialized_params=None):
        """Sends the initialized notification to LSP server."""
        if initialized_params is None:
            initialized_params = {}
        self._endpoint.notify("initialized", initialized_params)

    def shutdown(self, should_exit, exit_timeout: float = LSP_EXIT_TIMEOUT):
        """Sends the shutdown request to LSP server."""

        def _after_shutdown(_):
            if should_exit:
                self.exit_lsp(exit_timeout)

        self._send_request("shutdown", handle_response=_after_shutdown)

    def exit_lsp(self, exit_timeout: float = LSP_EXIT_TIMEOUT):
        """Handles LSP server process exit."""
        self._endpoint.notify("exit")
        assert unwrap(self._sub).wait(exit_timeout) == 0

    def notify_did_change(self, did_change_params):
        """Sends did change notification to LSP Server."""
        self._send_notification("textDocument/didChange", params=did_change_params)

    def notify_did_save(self, did_save_params):
        """Sends did save notification to LSP Server."""
        self._send_notification("textDocument/didSave", params=did_save_params)

    def notify_did_open(self, did_open_params):
        """Sends did open notification to LSP Server."""
        self._send_notification("textDocument/didOpen", params=did_open_params)

    def notify_did_close(self, did_close_params):
        """Sends did close notification to LSP Server."""
        self._send_notification("textDocument/didClose", params=did_close_params)

    def set_notification_callback(self, notification_name, callback):
        """Set custom LS notification handler."""
        self._notification_callbacks[notification_name] = callback

    def get_notification_callback(self, notification_name):
        """Gets callback if set or default callback for a given LS notification."""
        try:
            return self._notification_callbacks[notification_name]
        except KeyError:

            def _default_handler(_params):
                """Default notification handler."""

            return _default_handler

    def _publish_diagnostics(self, publish_diagnostics_params):
        """Internal handler for text document publish diagnostics."""
        return self._handle_notification(
            PUBLISH_DIAGNOSTICS, publish_diagnostics_params
        )

    def _window_log_message(self, window_log_message_params):
        """Internal handler for window log message."""
        return self._handle_notification(WINDOW_LOG_MESSAGE, window_log_message_params)

    def _window_show_message(self, window_show_message_params):
```

```
        """Internal handler for window show message."""
        return self._handle_notification(
            WINDOW_SHOW_MESSAGE, window_show_message_params
        )

    def _handle_notification(self, notification_name, params):
        """Internal handler for notifications."""
        fut: Future = Future()

        def _handler():
            callback = self.get_notification_callback(notification_name)
            callback(params)
            fut.set_result(None)

        self._thread_pool.submit(_handler)
        return fut

    def _send_request(self, name, params=None, handle_response=lambda f: f.done()):
        """Sends {name} request to the LSP server."""
        fut = self._endpoint.request(name, params)
        fut.add_done_callback(handle_response)
        return fut

    def _send_notification(self, name, params=None):
        """Sends {name} notification to the LSP server."""
        self._endpoint.notify(name, params)
}
```

tests/client/utils.py

```
"""Utility functions for use with tests."""

from __future__ import annotations

import pathlib
import platform
from typing import TypeVar


def normalizecase(path: str) -> str:
    """Fixes 'file' uri or path case for easier testing in windows."""
    if platform.system() == "Windows":
        return path.lower()
    return path


def as_uri(path: str) -> str:
    """Return 'file' uri as string."""
    return normalizecase(pathlib.Path(path).as_uri())


T = TypeVar("T")


def unwrap(option: T | None) -> T:
    if option is None:
        raise ValueError("Option is None")
```

```
    return option
}
```

tests/conftest.py

```
import subprocess
from pathlib import Path

import pytest
from packaging.version import Version

from ruff_lsp.server import Executable, _find_ruff_binary, _get_global_defaults, uris
from ruff_lsp.settings import WorkspaceSettings


def _get_ruff_executable() -> Executable:
    # Use the ruff-lsp directory as the workspace
    workspace_path = str(Path(__file__).parent.parent)

    settings = WorkspaceSettings(  # type: ignore[misc]
        **_get_global_defaults(),
        cwd=None,
        workspacePath=workspace_path,
        workspace=uris.from_fs_path(workspace_path),
    )

    return _find_ruff_binary(settings, version_requirement=None)


@pytest.fixture(scope="session")
def ruff_version() -> Version:
    return _get_ruff_executable().version


def pytest_report_header(config):
    """Add ruff version to pytest header."""
    executable = _get_ruff_executable()

    # Display the long version if the executable supports it
    try:
        output = subprocess.check_output([executable.path, "version"]).decode().strip()
    except subprocess.CalledProcessError:
        output = (
            subprocess.check_output([executable.path, "--version"]).decode().strip()
        )

    version = output.replace("ruff ", "")
    return [f"ruff-version: {version}"]
}
```

tests/test_format.py

```
from __future__ import annotations

from contextlib import nullcontext
```

```
import pytest
from lsprotocol.types import Position, Range
from packaging.version import Version
from pygls.workspace import Workspace

from ruff_lsp.server import (
    VERSION_REQUIREMENT_FORMATTER,
    VERSION_REQUIREMENT_RANGE_FORMATTING,
    Document,
    _fixed_source_to_edits,
    _get_settings_by_document,
    _run_format_on_document,
)
from tests.client import utils

original = """
x = 1
"""

expected = """x = 1
"""


@pytest.mark.asyncio
async def test_format(tmp_path, ruff_version: Version):
    test_file = tmp_path.joinpath("main.py")
    test_file.write_text(original)
    uri = utils.as_uri(str(test_file))

    workspace = Workspace(str(tmp_path))
    document = Document.from_text_document(workspace.get_text_document(uri))
    settings = _get_settings_by_document(document.path)

    handle_unsupported = (
        pytest.raises(RuntimeError, match=f"Ruff .* required, but
found {ruff_version}")
        if not VERSION_REQUIREMENT_FORMATTER.contains(ruff_version)
        else nullcontext()
    )

    with handle_unsupported:
        result = await _run_format_on_document(document, settings, None)
        assert result is not None
        assert result.exit_code == 0
        [edit] = _fixed_source_to_edits(
            original_source=document.source, fixed_source=result.stdout.decode("utf-8")
        )
        assert edit.new_text == ""
        assert edit.range == Range(
            start=Position(line=0, character=0), end=Position(line=1, character=0)
        )


@pytest.mark.asyncio
async def test_format_code_with_syntax_error(tmp_path, ruff_version: Version):
    source = """
foo =
"""

    test_file = tmp_path.joinpath("main.py")
    test_file.write_text(source)
    uri = utils.as_uri(str(test_file))
```

```
    workspace = Workspace(str(tmp_path))
    document = Document.from_text_document(workspace.get_text_document(uri))
    settings = _get_settings_by_document(document.path)

    handle_unsupported = (
        pytest.raises(RuntimeError, match=f"Ruff .* required, but
found {ruff_version}")
        if not VERSION_REQUIREMENT_FORMATTER.contains(ruff_version)
        else nullcontext()
    )

    with handle_unsupported:
        result = await _run_format_on_document(document, settings, None)
        assert result is not None
        assert result.exit_code == 2


@pytest.mark.asyncio
async def test_format_range(tmp_path, ruff_version: Version):
    original = """x   = 1



print( "Formatted")

print ("Not formatted")
"""

    expected = """print("Formatted")\n"""

    test_file = tmp_path.joinpath("main.py")
    test_file.write_text(original)
    uri = utils.as_uri(str(test_file))

    workspace = Workspace(str(tmp_path))
    document = Document.from_text_document(workspace.get_text_document(uri))
    settings = _get_settings_by_document(document.path)

    handle_unsupported = (
        pytest.raises(RuntimeError, match=f"Ruff .* required, but
found {ruff_version}")
        if not VERSION_REQUIREMENT_RANGE_FORMATTING.contains(ruff_version)
        else nullcontext()
    )

    with handle_unsupported:
        result = await _run_format_on_document(
            document,
            settings,
            Range(
                start=Position(line=1, character=0),
                end=(Position(line=4, character=19)),
            ),
        )
        assert result is not None
        assert result.exit_code == 0
        [edit] = _fixed_source_to_edits(
            original_source=document.source, fixed_source=result.stdout.decode("utf-8")
        )
        assert edit.new_text == expected
        assert edit.range == Range(
            start=Position(line=3, character=0), end=Position(line=5, character=0)
        )
}
```

tests/test_server.py

```
"""Test for linting over LSP."""

from __future__ import annotations

import os
import tempfile
from dataclasses import dataclass
from threading import Event

from packaging.version import Version
from typing_extensions import Self

from tests.client import defaults, session, utils

# Increase this if you want to attach a debugger
TIMEOUT_SECONDS = 10

CONTENTS = """import sys

print(x)
"""

VERSION_REQUIREMENT_ASTRAL_DOCS = Version("0.0.291")
VERSION_REQUIREMENT_APPLICABILITY_RENAME = Version("0.1.0")


@dataclass(frozen=True)
class Applicability:
    safe: str
    unsafe: str
    display: str

    @classmethod
    def from_ruff_version(cls, ruff_version: Version) -> Self:
        if ruff_version >= VERSION_REQUIREMENT_APPLICABILITY_RENAME:
            return cls(safe="safe", unsafe="unsafe", display="display")
        else:
            return cls(safe="Automatic", unsafe="Suggested", display="Manual")


class TestServer:
    maxDiff = None

    def test_linting_example(self, ruff_version: Version) -> None:
        expected_docs_url = (
            "https://docs.astral.sh/ruff/"
            if ruff_version >= VERSION_REQUIREMENT_ASTRAL_DOCS
            else "https://beta.ruff.rs/docs/"
        )
        applicability = Applicability.from_ruff_version(ruff_version)

        with tempfile.NamedTemporaryFile(suffix=".py") as fp:
            fp.write(CONTENTS.encode())
            fp.flush()
            uri = utils.as_uri(fp.name)

            actual = []
```

```
        with session.LspSession(cwd=os.getcwd(), module="ruff_lsp") as ls_session:
            ls_session.initialize(defaults.VSCODE_DEFAULT_INITIALIZE)

            done = Event()

            def _handler(params):
                nonlocal actual
                actual = params
                done.set()

            ls_session.set_notification_callback(
                session.PUBLISH_DIAGNOSTICS, _handler
            )

            ls_session.notify_did_open(
                {
                    "textDocument": {
                        "uri": uri,
                        "languageId": "python",
                        "version": 1,
                        "text": CONTENTS,
                    }
                }
            )

            # Wait to receive all notifications.
            done.wait(TIMEOUT_SECONDS)

            expected = {
                "diagnostics": [
                    {
                        "code": "F401",
                        "codeDescription": {
                            "href": expected_docs_url + "rules/unused-import"
                        },
                        "data": {
                            "fix": {
                                "applicability": applicability.safe,
                                "edits": [
                                    {
                                        "content": "",
                                        "end_location": {"column": 0, "row": 2},
                                        "location": {"column": 0, "row": 1},
                                    }
                                ],
                                "message": "Remove unused import: `sys`",
                            },
                            "noqa_row": 1,
                            "cell": None,
                        },
                        "message": "`sys` imported but unused",
                        "range": {
                            "end": {"character": 10, "line": 0},
                            "start": {"character": 7, "line": 0},
                        },
                        "severity": 2,
                        "source": "Ruff",
                        "tags": [1],
                    },
                    {
                        "code": "F821",
                        "codeDescription": {
                            "href": expected_docs_url + "rules/undefined-name"
                        },
```

```
                        "data": {"fix": None, "noqa_row": 3, "cell": None},
                        "message": "Undefined name `x`",
                        "range": {
                            "end": {"character": 7, "line": 2},
                            "start": {"character": 6, "line": 2},
                        },
                        "severity": 1,
                        "source": "Ruff",
                    },
                ],
                "uri": uri,
            }
        assert expected == actual

def test_no_initialization_options(self, ruff_version: Version) -> None:
    expected_docs_url = (
        "https://docs.astral.sh/ruff/"
        if ruff_version >= VERSION_REQUIREMENT_ASTRAL_DOCS
        else "https://beta.ruff.rs/docs/"
    )
    applicability = Applicability.from_ruff_version(ruff_version)

    with tempfile.NamedTemporaryFile(suffix=".py") as fp:
        fp.write(CONTENTS.encode())
        fp.flush()
        uri = utils.as_uri(fp.name)

        actual = []
        with session.LspSession(cwd=os.getcwd(), module="ruff_lsp") as ls_session:
            ls_session.initialize(
                {
                    **defaults.VSCODE_DEFAULT_INITIALIZE,
                    "initializationOptions": None,
                }
            )

            done = Event()

            def _handler(params):
                nonlocal actual
                actual = params
                done.set()

            ls_session.set_notification_callback(
                session.PUBLISH_DIAGNOSTICS, _handler
            )

            ls_session.notify_did_open(
                {
                    "textDocument": {
                        "uri": uri,
                        "languageId": "python",
                        "version": 1,
                        "text": CONTENTS,
                    }
                }
            )

            # Wait to receive all notifications.
            done.wait(TIMEOUT_SECONDS)

            expected = {
                "diagnostics": [
                    {
```

```
                        "code": "F401",
                        "codeDescription": {
                            "href": expected_docs_url + "rules/unused-import"
                        },
                        "data": {
                            "fix": {
                                "applicability": applicability.safe,
                                "edits": [
                                    {
                                        "content": "",
                                        "end_location": {"column": 0, "row": 2},
                                        "location": {"column": 0, "row": 1},
                                    }
                                ],
                                "message": "Remove unused import: `sys`",
                            },
                            "noqa_row": 1,
                            "cell": None,
                        },
                        "message": "`sys` imported but unused",
                        "range": {
                            "end": {"character": 10, "line": 0},
                            "start": {"character": 7, "line": 0},
                        },
                        "severity": 2,
                        "source": "Ruff",
                        "tags": [1],
                    },
                    {
                        "code": "F821",
                        "codeDescription": {
                            "href": expected_docs_url + "rules/undefined-name"
                        },
                        "data": {"fix": None, "noqa_row": 3, "cell": None},
                        "message": "Undefined name `x`",
                        "range": {
                            "end": {"character": 7, "line": 2},
                            "start": {"character": 6, "line": 2},
                        },
                        "severity": 1,
                        "source": "Ruff",
                    },
                ],
                "uri": uri,
            }
        assert expected == actual
}
```

The End

This PDF was generated by gitprint.me