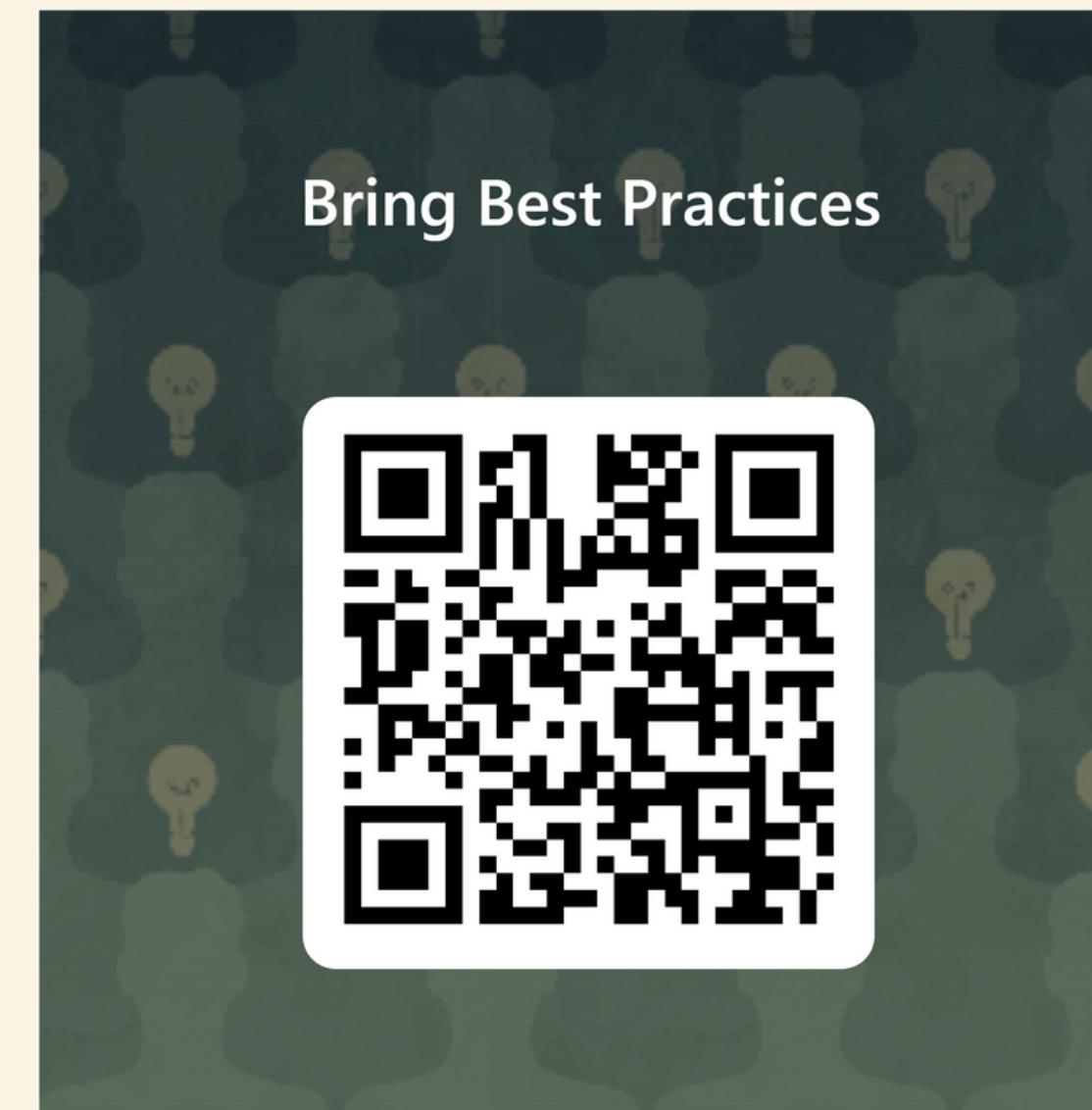

welcome to

BRING BEST PRACTICES TO YOUR MESSY DATA SCIENCE TEAM!

with Reproducibility, Compliance, and Consistency in Mind

Live
Pod



FORMS.OFFICE.COM/R/V8VUEWPOBP

introduce **ABOUT ME**

A husband, a dad for 6yo and 2yo girls, a teacher, an archer and mountain biker

PhD from King's College London in Medical Imaging Sciences in 2015

Lead Data Scientist and **MIOps**
at NIQ BrandBank



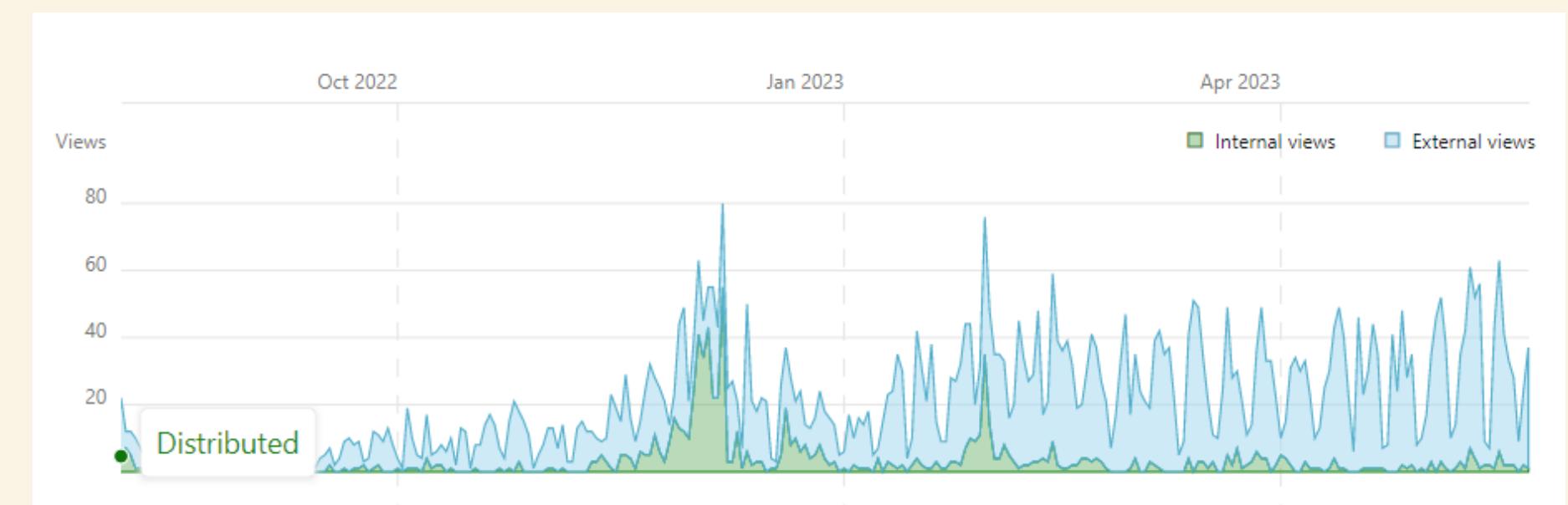
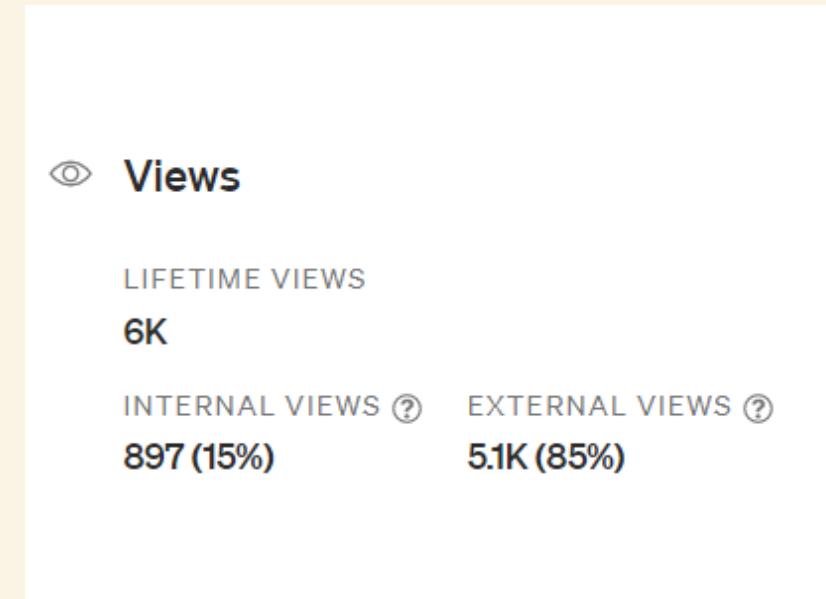
**GABRIEL
HARRIS**



Python HOW: Using Poetry, Make, and pre-commit-hooks to Setup a Repo Template for your Team

Bring consistency, rigour, and best practices to your messy data science team

Medium / Aug 5, 2022



why?

-_(ツ)_/-
IT WORKS
on my machine

REPRODUCIBILITY

why?

IT'S LIKE SOMEONE TOOK A
TRANSCRIPT OF A COUPLE
ARGUING AT IKEA AND MADE
RANDOM EDITS UNTIL IT
COMPILED WITHOUT ERRORS.



COMPLIANCE

why?

EVENTUAL CONSISTENCY IS
GUARANTEED BY THE 2ND
LAW OF THERMODYNAMICS.

SOONER OR LATER
THIS WILL ALL BE A
UNIFORM HEAT BATH.

MAXIMUM
ENTROPY.



CONSISTENCY

how?

PIP-TOOLS



REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS



COMPLIANCE

how?

GNU MAKE



CONSISTENCY

Breathe

PRE PROJECT

HELPER PACKAGES

BLACK

ISORT

PYDOCSTYLE

RUFF

SQLFLUFF

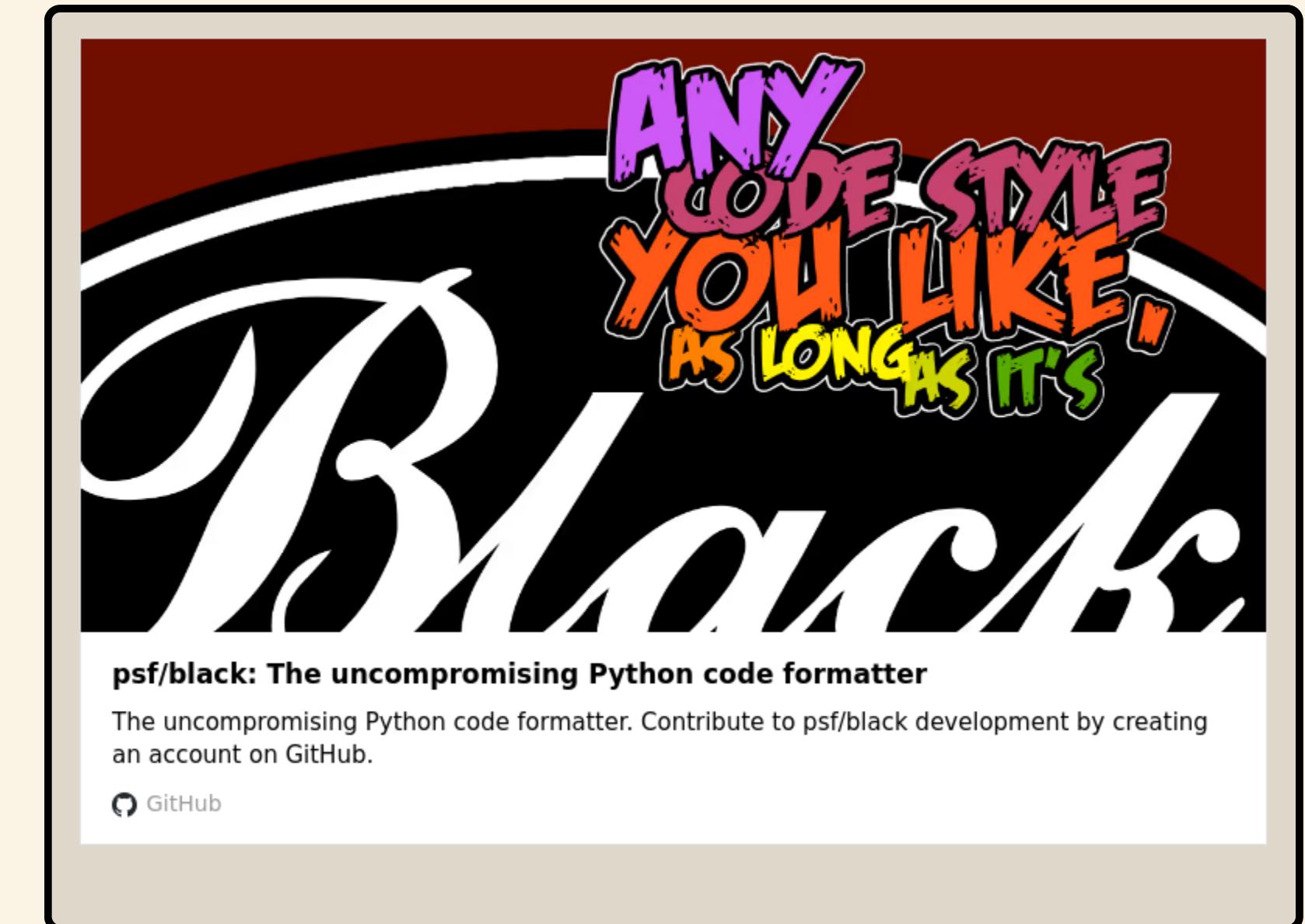
PRE-COMMIT

BLACK

30.4K

Stars

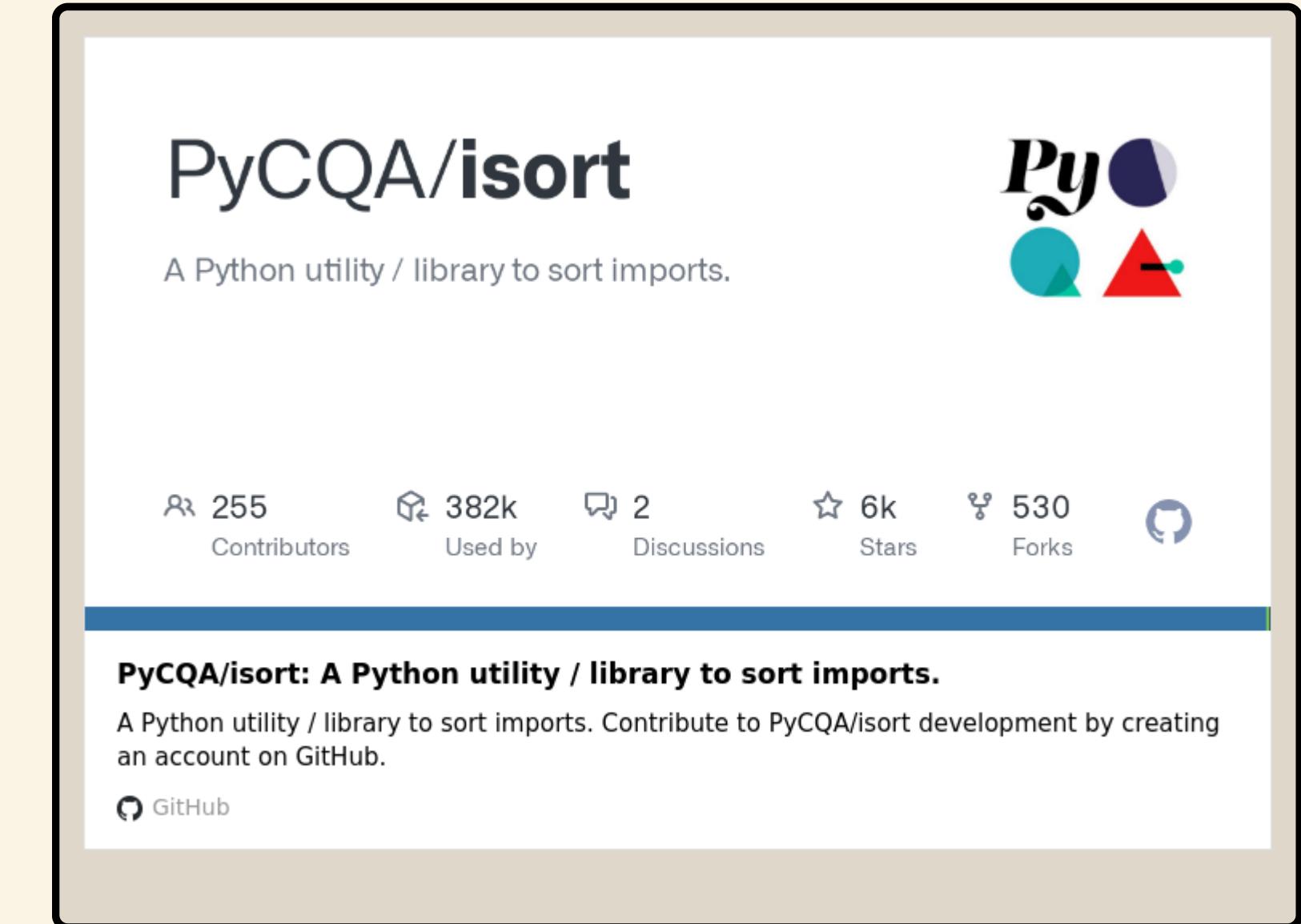
"Black is the uncompromising Python code formatter. By using it, you agree to cede control over minutiae of hand-formatting. In return, Black gives you speed, determinism, and freedom from **pycodestyle** nagging about formatting. You will save time and mental energy for more important matters."



ISORT

5.8K
Stars

"isort sort imports alphabetically, and automatically separated into sections and by type."

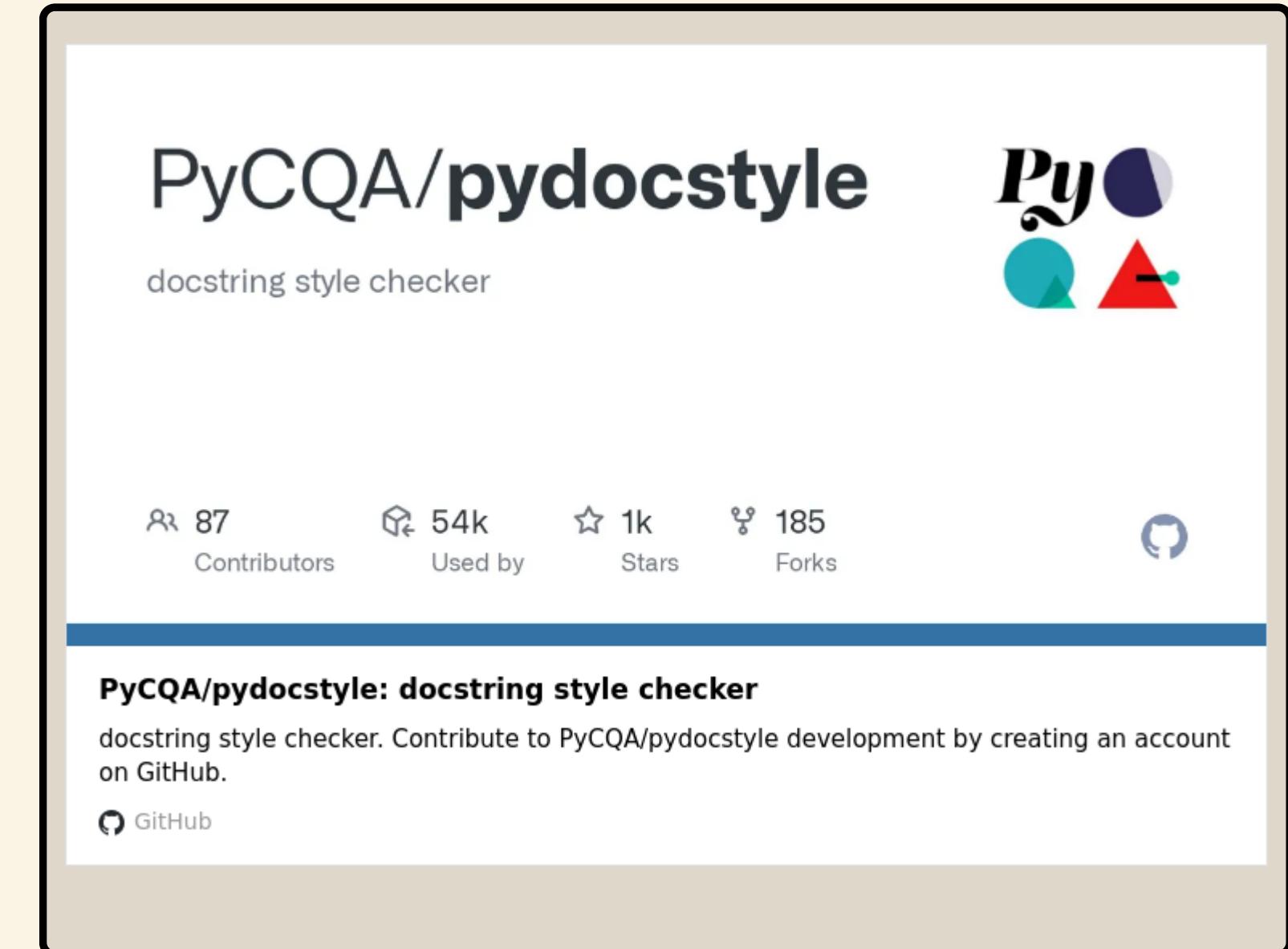


PYDOCSTYLE

1K Stars

"pydocstyle is a static analysis tool for checking compliance with Python docstring conventions."

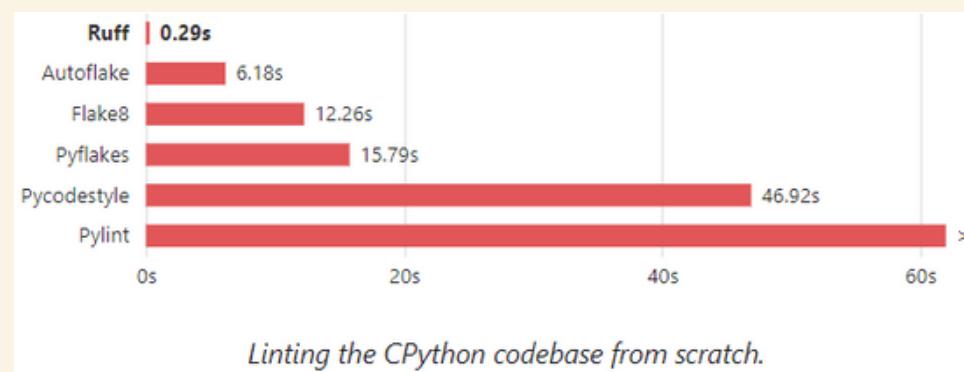
"There are three conventions that may be used by pydocstyle: **pep257**, **numpy** and **google**".



RUFF

15K
Stars

An extremely fast Python linter, written in **Rust**.
Can be used to replace Flake8, isort, pydocstyle
(plus dozens of plugins) all while executing tens
or hundreds of times faster than any individual
tool



The screenshot shows the GitHub repository page for `charliermarsh/ruff`. The page includes a profile picture of the maintainer, a brief description, and various statistics: 199 contributors, 6k used by, 51 discussions, 15k stars, 450 forks, and a GitHub icon.

charliermarsh/ruff

An extremely fast Python linter, written in Rust.

199 Contributors 6k Used by 51 Discussions 15k Stars 450 Forks

charliermarsh/ruff: An extremely fast Python linter, written in Rust.

An extremely fast Python linter, written in Rust. Contribute to charliermarsh/ruff development by creating an account on GitHub.

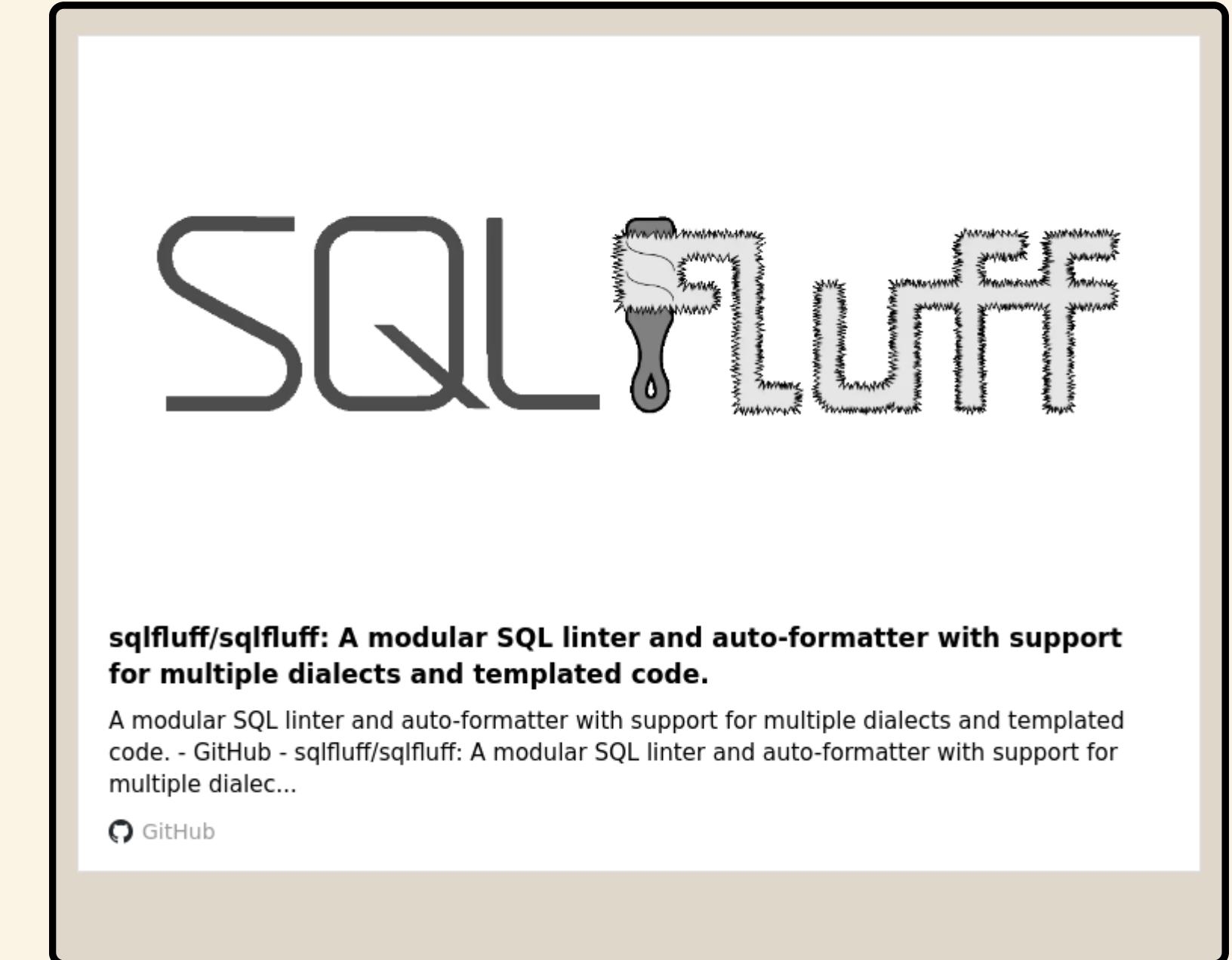
[GitHub](#)

SQLFLUFF

6.1K

Stars

"SQLFluff is a dialect-flexible and configurable SQL linter. Designed with ELT applications in mind, SQLFluff also works with Jinja templating and dbt. SQLFluff will auto-fix most linting errors, allowing you to focus your time on what matters."

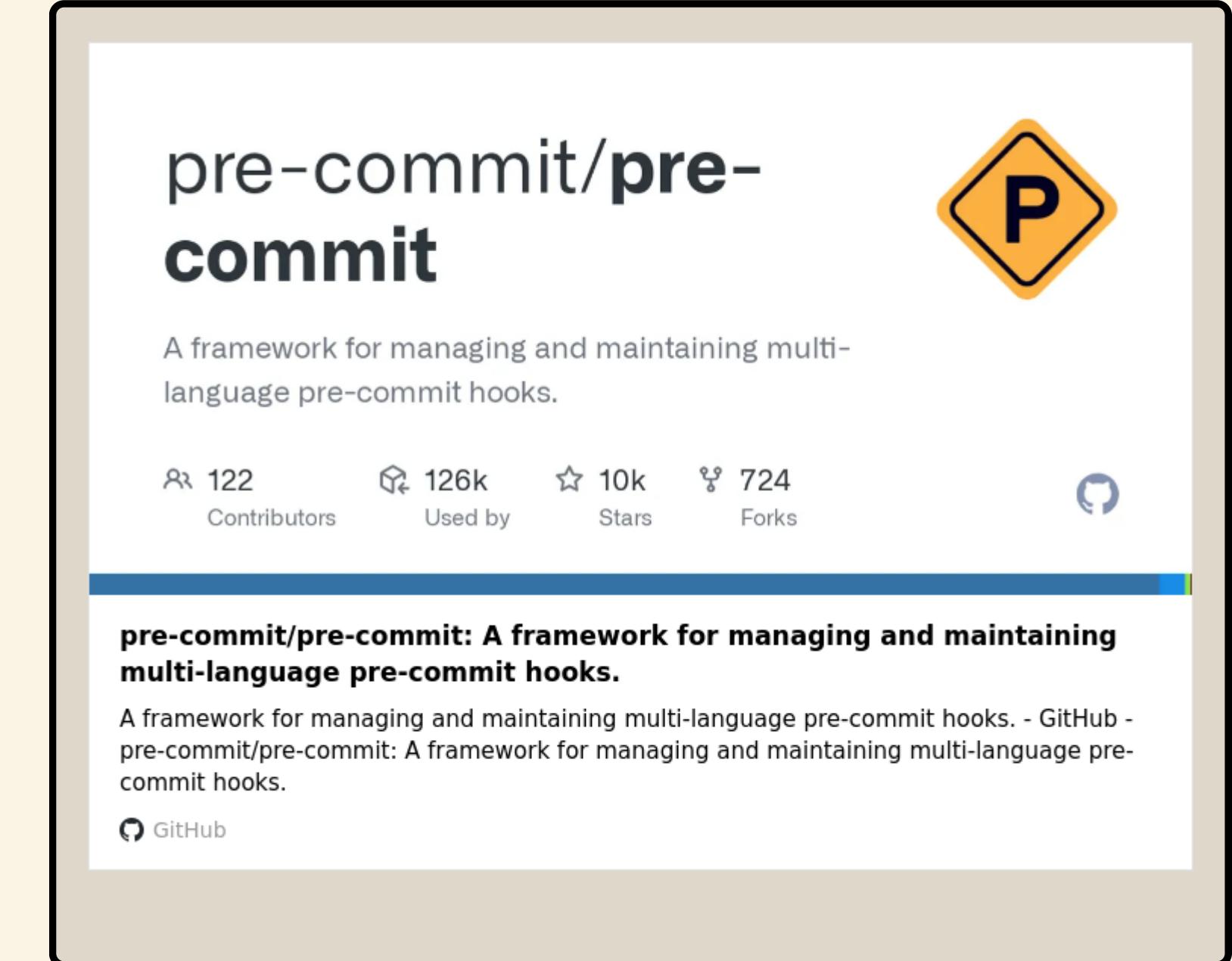


PRE-COMMIT

10.4K

Stars

"A framework for managing and maintaining multi-language pre-commit hooks."



Breathe

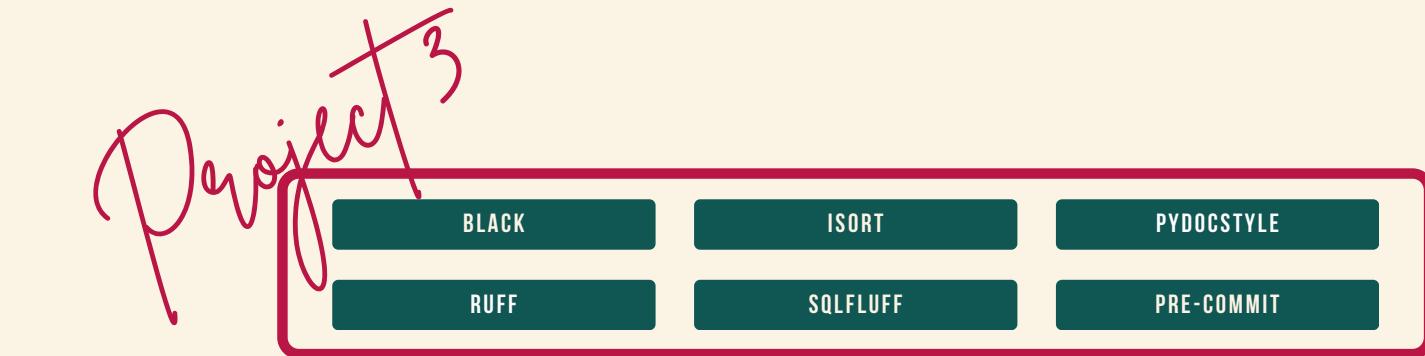
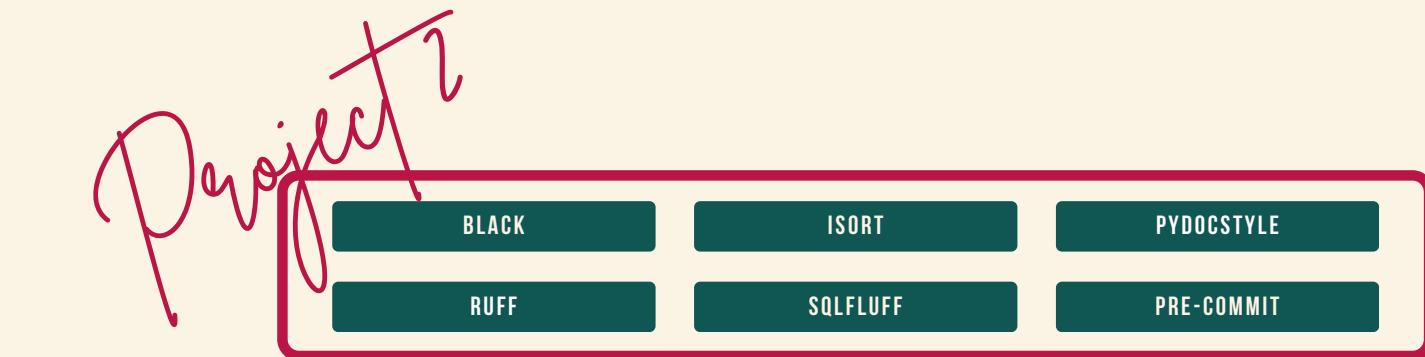
HELPER PACKAGES

INSTALLATION

HELPER PACKAGES PER PROJECT

"Install packages for each project."

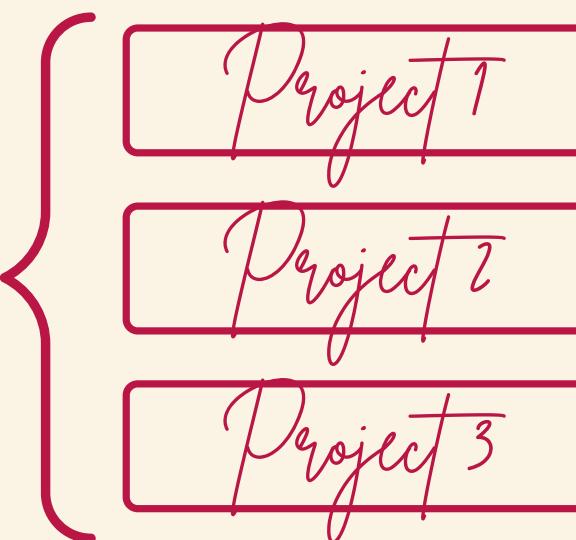
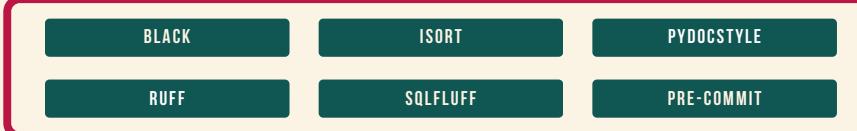
We can do better



HELPER PACKAGES WITH PIPX

6.5K
Stars

"**PIPX** installs packages in isolated environments that can be run globally."



The screenshot shows the GitHub profile for the repository "pypa/pipx". The repository has 89 contributors, 1k used by projects, 6k stars, and 297 forks. The description reads: "Install and Run Python Applications in Isolated Environments". Below the description is a link to the GitHub repository: "pypa/pipx: Install and Run Python Applications in Isolated Environments".

HELPER PACKAGES WITH PIPX

Create virtual environment for pipx

- It's best to create a virtual environment just for pipx in your \$HOME\.venv
- It doesn't matter which python version you choose. We will use 3.11.

```
> py -3.11 -m venv $HOME\.venv\pipx.venv
```

HELPER PACKAGES WITH PIPX

Install pipx

- Activate pipx.venv and install pipx

```
> $HOME\.venv\pipx.venv\Scripts\Activate.ps1  
(pipx.venv) > pip install pipx
```

- This installs pipx.exe in \$HOME \.venv \pipx.venv \Scripts which is not in \$PATH
- To fix this, add it to \$PATH and check it works

```
(pipx.venv) > pipx --version  
1.2.0
```

HELPER PACKAGES WITH PIPX

Install helper packages

```
(pipx.venv) > pipx install black  
(pipx.venv) > pipx install isort  
(pipx.venv) > pipx install pydocstyle  
(pipx.venv) > pipx install ruff  
(pipx.venv) > pipx install sqlfluff  
(pipx.venv) > pipx install pre-commit
```

- pipx installs package executables in \$HOME\.local\bin which is not in \$PATH
- To fix this, add it to \$PATH , or run pipx ensurepath

```
(pipx.venv) > pipx ensurepath  
Success! Added C:\Users\<username>\.local\bin to the PATH environment variable.  
You will need to open a new terminal or re-login for the PATH changes to take effect.  
Otherwise pipx is ready to go! ✨ ✨
```

HELPER PACKAGES WITH PIPX

List installed packages

```
> pipx list
venvs are in C:\Users\<username>\.local\pipx\venvs
apps are exposed on your $PATH at C:\Users\<username>\.local\bin
  package black 23.3.0, installed using Python 3.11.3
    - black.exe
    - blackd.exe
  package flake8 6.0.0, installed using Python 3.11.3
    - flake8.exe
  package isort 5.12.0, installed using Python 3.11.3
    - isort-identify-imports.exe
    - isort.exe
  package pip-chill 1.0.3, installed using Python 3.11.3
    - pip-chill.exe
  package pre-commit 3.3.1, installed using Python 3.11.3
    - pre-commit.exe
  package pydocstyle 6.3.0, installed using Python 3.11.3
    - pydocstyle.exe
  package ruff 0.0.269, installed using Python 3.11.3
    - ruff.exe
  package sqlfluff 2.0.7, installed using Python 3.11.3
    - sqlfluff.exe
```

HELPER PACKAGES

CONFIGURATIONS

HELPER PACKAGES WITH PYPROJECT.TOML



What the heck is pyproject.toml?

Recently on Twitter there was a maintainer of a Python project who had a couple of bugs filed against their project due to builds failing (this particular...

 Brett / Apr 1, 2020

"the purpose of [PEP 518](#) was to come up with a way for projects to specify what build tools they required ... all projects should (eventually) have ... development tools realized they ... could put their own configuration"

- Brett Cannon, Python Steering Council

```
[T] pyproject.toml ×  
[T] pyproject.toml  
22  
23 [tool.black]  
24 line-length = 88  
25  
26 [tool.isort]  
27 profile = "black"  
28  
29 [tool.pydocstyle]  
30 convention = "google"  
31 add-ignore = "D100,D101,D102,D103,D104,D105,D106,D107"  
32  
33 [tool.ruff]  
34 line-length = 88  
35  
36 [tool.sqlfluff.core]  
37 dialect = "tsql"  
38 exclude_rules = "L016,L031"  
39 max_line_length = 88
```

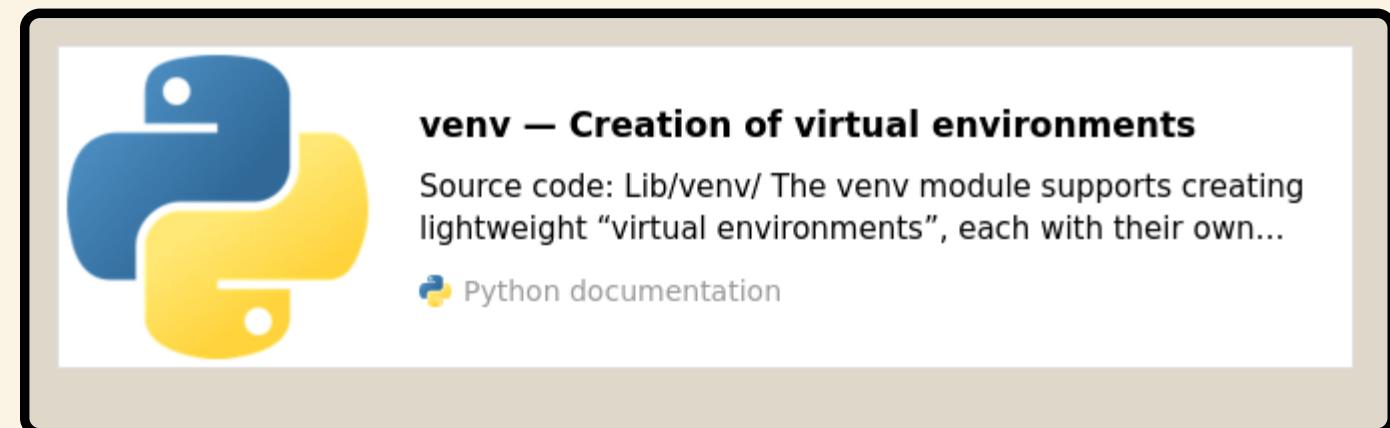
Breathe

NEW PROJECT

VENV MODULE

The **venv** module, proposed in [PEP 405](#), supports creating lightweight “virtual environments”, each with their own independent set of Python packages installed in their [site](#) directories.

You do isolate packages, right? right?



```
> py -3.10 -m venv $path\to\project\.venv  
> $path\to\project\.venv\Scripts\Activate.ps1  
(.venv) >
```

how?

PIP-TOOLS



REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS



COMPLIANCE

how?

GNU MAKE



CONSISTENCY

PIP-TOOLS

6.9K

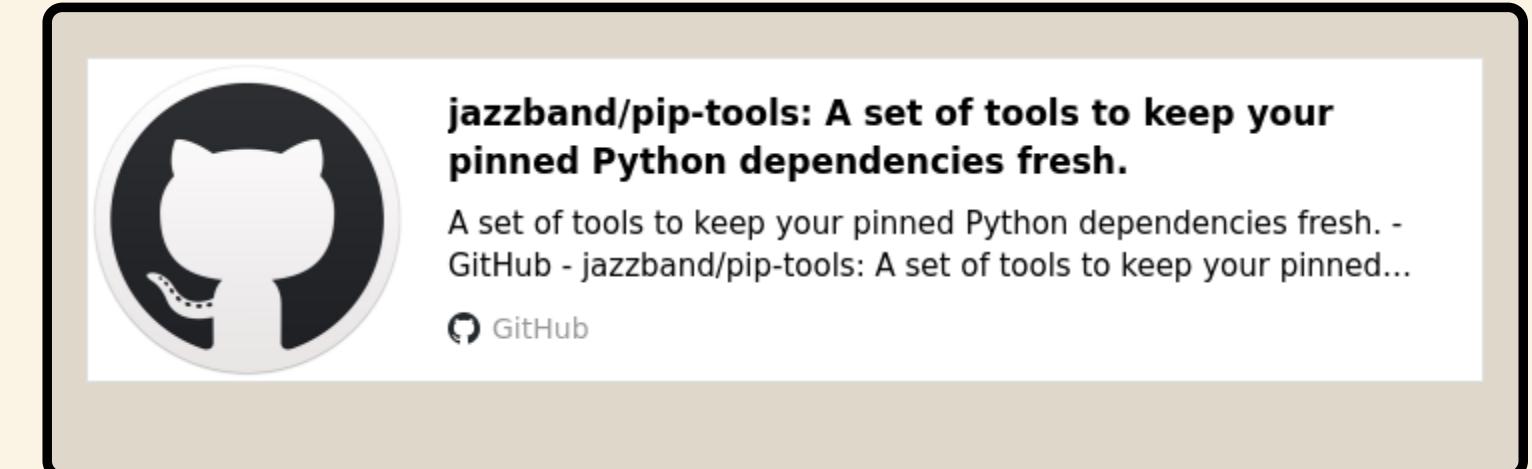
Stars

"A set of command line tools to help you keep your pip-based packages fresh, even when you've pinned them."

You do pin packages, right? right?

```
(.venv) > pip install pip-tools
```

Installed in each Project

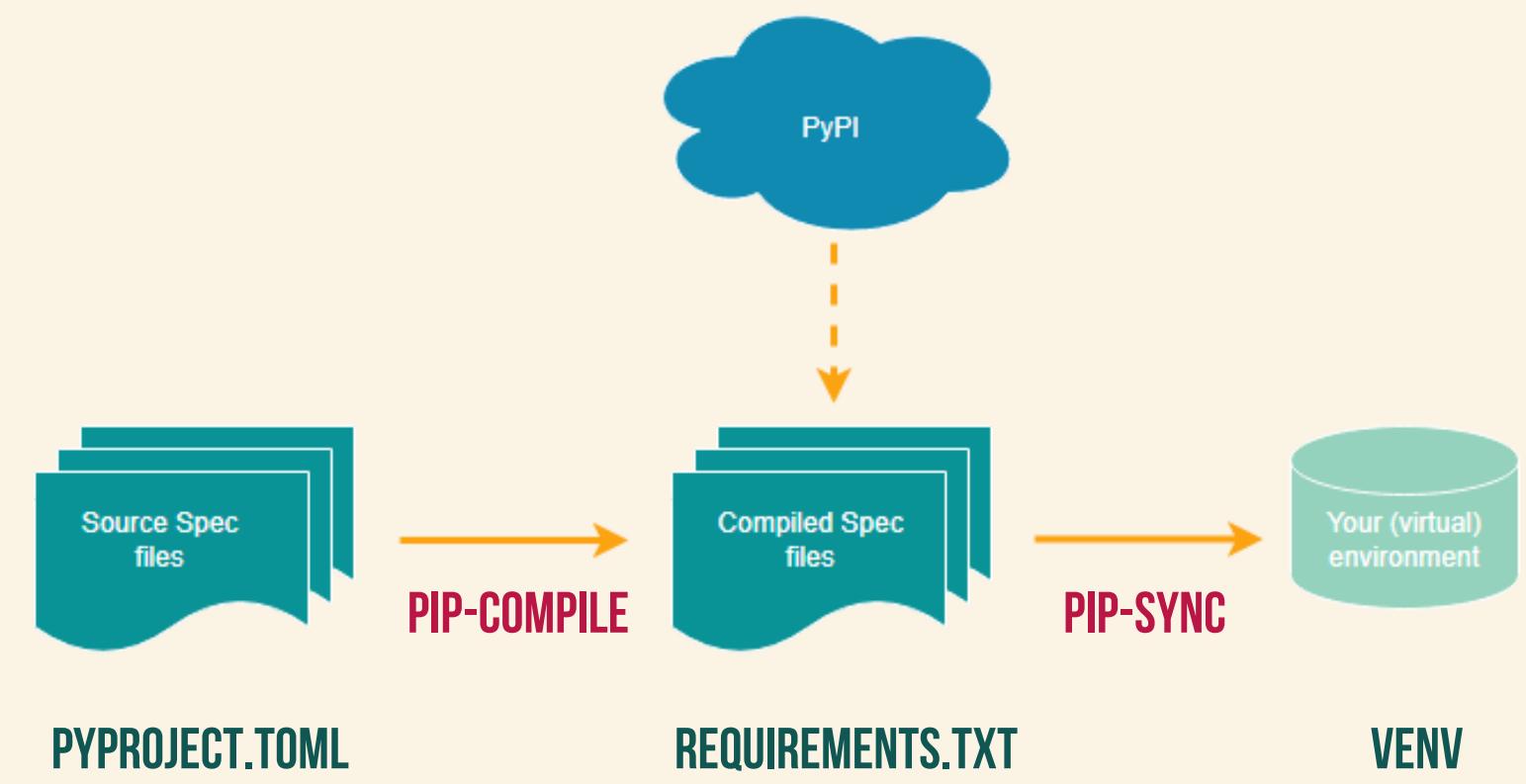


PIP-TOOLS =

PIP-COMPILE + PIP-SYNC

The **PIP-COMPILE** command compiles a **requirements.txt** file from your dependencies, specified in either **pyproject.toml**, setup.cfg, setup.py, or requirements.in

The **PIP-SYNC** command syncs your **venv** to reflect exactly what's in **requirements.txt**. This will install/upgrade/uninstall everything necessary to match requirements.txt contents



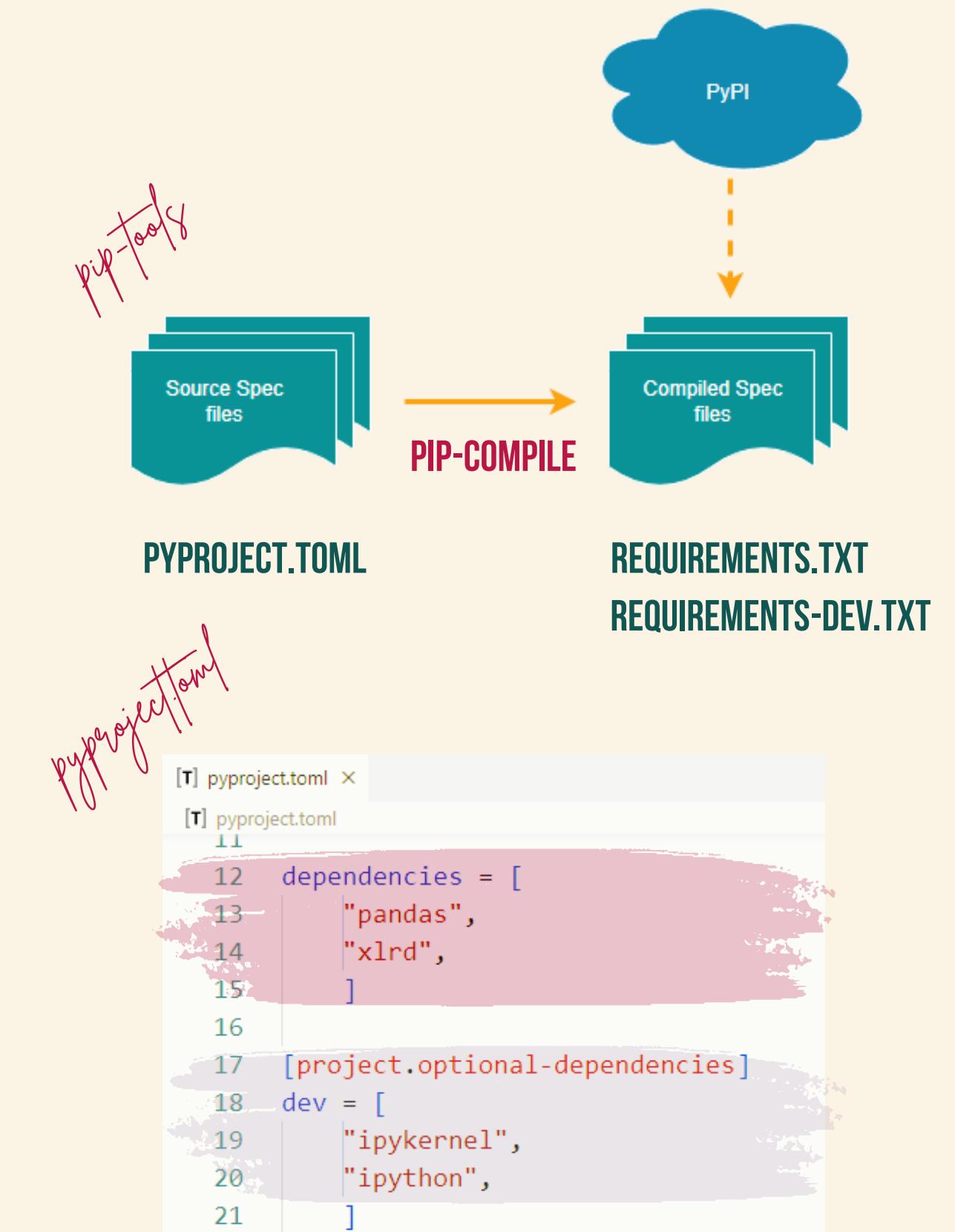
PIP-COMPILE

Generates **requirements.txt** using the latest versions that fulfil the **dependencies** specified in **pyproject.toml**

```
(.venv) > pip-compile pyproject.toml --output-file requirements.txt
```

Generates **requirements-dev.txt** using the latest versions that fulfil **dependencies + optional-dependencies** specified in **pyproject.toml**

```
(.venv) > pip-compile pyproject.toml --output-file requirements-dev.txt --extra dev
```



PIP-COMPILE

If an existing **requirements.txt** file is found that fulfils the dependencies, then no changes will be made.

To force **pip-compile** to update all packages in **requirements.txt**

```
(.venv) > pip-compile --upgrade
```

To update a specific package to the latest or a specific version

```
(.venv) > pip-compile --upgrade-package pandas  
(.venv) > pip-compile --upgrade-package pandas==2.0.0
```

Combine both to provide constraints on the allowed upgrades

```
(.venv) > pip-compile --upgrade --upgrade-package pandas==2.0.0
```

PIP-COMPIL

REQUIREMENTS.TXT

```
requirements.txt x
requirements.txt
1 #
2 # This file is autogenerated by pip-compile with Python 3.10
3 # by the following command:
4 #
5 #     pip-compile --allow-unsafe --output-file=requirements.txt --resolver=backtracking pyproject.toml
6 #
7 numpy==1.24.3
8     # via pandas
9 pandas==2.0.2
10    # via Template (pyproject.toml)
11 python-dateutil==2.8.2
12    # via pandas
13 pytz==2023.3
14    # via pandas
15 six==1.16.0
16    # via python-dateutil
17 tzdata==2023.3
18    # via pandas
19 xlrd==2.0.1
20    # via Template (pyproject.toml)
```

dependencies

REQUIREMENTS-DEV.TXT

```
requirements-dev.txt x
requirements-dev.txt
1 #
2 # This file is autogenerated by pip-compile with Python 3.10
3 # by the following command:
4 #
5 #     pip-compile --allow-unsafe --extra=dev --output-file=requirements-dev.txt --resolver=backtracking pyproject.toml
6 #
7 asttokens==2.2.1
8     # via stack-data
9 backcall==0.2.0
10    # via ipython
11 colorama==0.4.6
12    # via ipython
13 comm==0.1.3
14    # via ipykernel
15 debugpy==1.6.7
16    # via ipykernel
17 decorator==5.1.1
18    # via ipython
19 executing==1.2.0
20    # via stack-data
21 ipykernel==6.23.1
22    # via Template (pyproject.toml)
23 ipython==8.13.2
24    # via
25    #   - via Template (pyproject.toml)
```

dependencies + optional dependencies

PYPROJECT.TOML

```
[T] pyproject.toml x
[T] pyproject.toml
11
12 dependencies = [
13     "pandas",
14     "xlrd",
15 ]
16
17 [project.optional-dependencies]
18 dev = [
19     "ipykernel",
20     "ipython",
21 ]
```

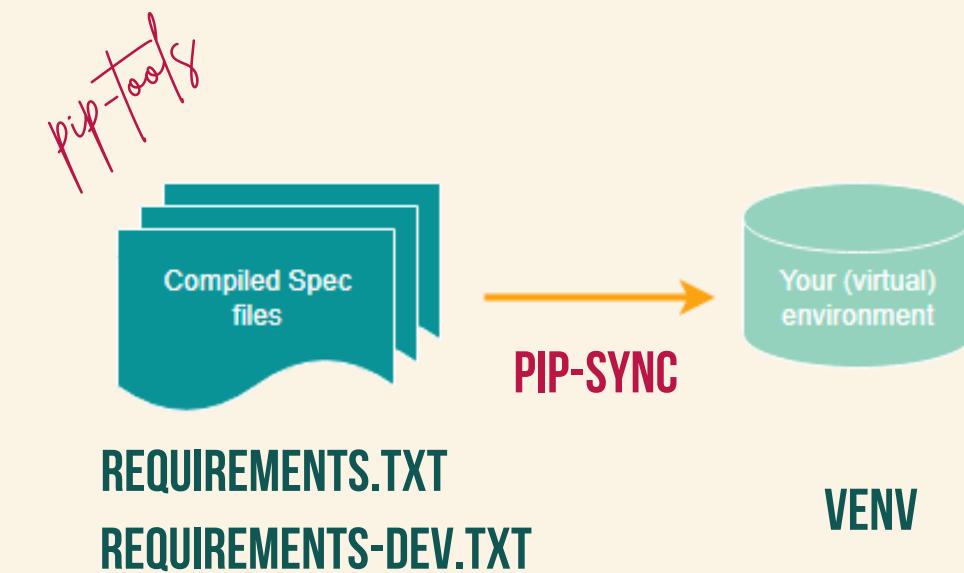
PIP-SYNC

Update **venv** to reflect exactly what's in **requirements.txt**. This will install, upgrade, and uninstall everything necessary

```
(.venv) > pip-sync requirements.txt
```

Or what's in **requirements-dev.txt**

```
(.venv) > pip-sync requirements-dev.txt
```



Breathe

how?

PIP-TOOLS

JAZZ
BAND

REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS



COMPLIANCE

how?

GNU MAKE



CONSISTENCY

PRE-COMMIT-HOOKS

"Git hook scripts are useful for identifying simple issues before submission to code review. By pointing these issues out before code review, this allows a code reviewer to focus on the architecture of a change while not wasting time with trivial style nitpicks."

Install pre-commit package



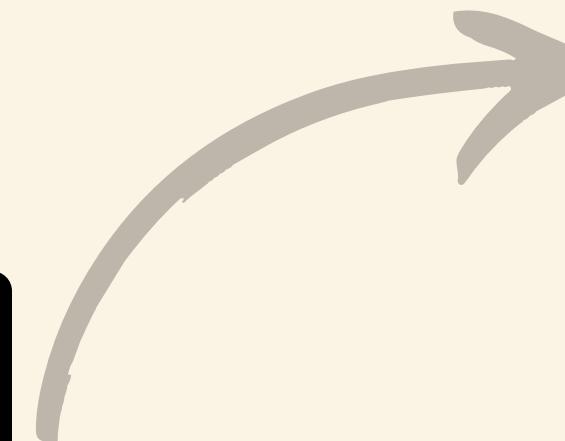
PRE-COMMIT-HOOKS

Add pre-commit configuration

Specify hook in `.pre-commit-config.yaml`

X Hook

```
- repo: <link to a git package with a .pre-commit-hooks.yaml>
  rev: <package version>
  hooks:
    - id: hook id found in .pre-commit-hooks.yaml
      args: list of parameters to pass to the hook (default [])
```



.PRE-COMMIT-CONFIG.YAML

```
7  .pre-commit-config.yaml ×
8  .pre-commit-config.yaml
9
10 - repo: https://github.com/PyCQA/isort
11   rev: '5.12.0'
12   hooks:
13     - id: isort
14       args: ['--profile=black']
```

PRE-COMMIT-HOOKS

BLACK

.PRE-COMMIT-CONFIG.YAML

```
- repo: https://github.com/psf/black
  rev: '23.3.0'
  hooks:
    - id: black
    - id: black-jupyter
```

BLACK/.PRE-COMMIT-HOOKS.YAML

```
1 - id: black
2   name: black
3   description: "Black: The uncompromising Python code formatter"
4   entry: black
5   language: python
6   minimum_pre_commit_version: 2.9.2
7   require_serial: true
8   types_or: [python, pyi]
9 - id: black-jupyter
10  name: black-jupyter
11  description:
12    "Black: The uncompromising Python code formatter (with Jupyter Notebook support)"
13  entry: black
14  language: python
15  minimum_pre_commit_version: 2.9.2
16  require_serial: true
17  types_or: [python, pyi, jupyter]
18  additional_dependencies: [".[jupyter]"]
```

PRE-COMMIT-HOOKS

PYDOCSTYLE

.PRE-COMMIT-CONFIG.YAML

```
- repo: https://github.com/pycqa/pydocstyle
  rev: '6.3.0'
  hooks:
    - id: pydocstyle
    - args: ['--add-ignore=D100,D101,D102,D103,D104,D105,D106,D107',
      '--convention=google']
```

PYDOCSTYLE/.PRE-COMMIT-HOOKS.YAML

```
1   id: pydocstyle
2   name: pydocstyle
3   description: pydocstyle is a static analysis tool for checking compliance with Python docstring conventions.
4   entry: pydocstyle
5   language: python
6   types: [python]
```

PRE-COMMIT-HOOKS

RUFF

.PRE-COMMIT-CONFIG.YAML

```
- repo: https://github.com/charliermarsh/ruff-pre-commit
  rev: 'v0.0.270'
  hooks:
    - id: ruff
```

RUFF-PRE-COMMIT/.PRE-COMMIT-HOOKS.YAML

```
1 - id: ruff
2   name: ruff
3   description: "Run 'ruff' for extremely fast Python linting"
4   entry: ruff check --force-exclude
5   language: python
6   'types_or': [python, pyi]
7   args: []
8   require_serial: true
9   additional_dependencies: []
10  minimum_pre_commit_version: '2.9.2'
```

PRE-COMMIT-HOOKS

SQLFLUFF

.PRE-COMMIT-CONFIG.YAML

```
- repo: https://github.com/sqlfluff/sqlfluff
  rev: '2.1.1'
  hooks:
    - id: sqlfluff-fix
    - args: ['--dialect=tsql', '--exclude_rules=L016,L031']
    - id: sqlfluff-lint
    - args: ['--dialect=tsql', '--exclude_rules=L016,L031']
```

SQLFLUFF/.PRE-COMMIT-HOOKS.YAML

```
1  - id: sqlfluff-lint
2    name: sqlfluff-lint
3    # Set `--processes 0` to use maximum parallelism
4    entry: sqlfluff lint --processes 0
5    language: python
6    description: "Lints sql files with `SQLFluff`"
7    types: [sql]
8    require_serial: true
9    additional_dependencies: []
10
11 - id: sqlfluff-fix
12   name: sqlfluff-fix
13   # Set a couple of default flags:
14   # - `--force` to disable confirmation
15   # - `--show-lint-violations` shows issues to not require running `sqlfluff lint`
16   # - `--processes 0` to use maximum parallelism
17   # By default, this hook applies all rules.
18   entry: sqlfluff fix --force --show-lint-violations --processes 0
19   language: python
20   description: "Fixes sql lint errors with `SQLFluff`"
21   types: [sql]
22   require_serial: true
23   additional_dependencies: []
```

PRE-COMMIT-HOOKS

PRE-COMMIT-HOOKS

.PRE-COMMIT-CONFIG.YAML

```
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: 'v4.4.0'
  hooks:
    - id: check-ast
    - id: check-json
    - id: check-toml
    - id: check-yaml
    - id: end-of-file-fixer
    - id: trailing-whitespace
```

PRE-COMMIT-HOOKS/.PRE-COMMIT-HOOKS.YAML

```
7 - id: check-ast
8   name: check python ast
9   description: simply checks whether the files parse as valid python.
10  entry: check-ast
11  language: python
12  types: [python]

13 - id: check-json
14   name: check json
15   description: checks json files for parseable syntax.
16   entry: check-json
17   language: python
18   types: [json]

19 - id: check-toml
20   name: check toml
21   description: checks toml files for parseable syntax.
22   entry: check-toml
23   language: python
24   types: [toml]

25 - id: check-yaml
26   name: check yaml
27   description: checks yaml files for parseable syntax.
28   entry: check-yaml
29   language: python
30   types: [yaml]

31 - id: end-of-file-fixer
32   name: fix end of files
33   description: ensures that a file is either empty, or ends with one newline.
34   entry: end-of-file-fixer
35   language: python
36   types: [text]
37   stages: [commit, push, manual]

38 - id: trailing whitespace
39   name: trim trailing whitespace
40   description: trims trailing whitespace.
41   entry: trailing-whitespace-fixer
42   language: python
43   types: [text]
44   stages: [commit, push, manual]
```

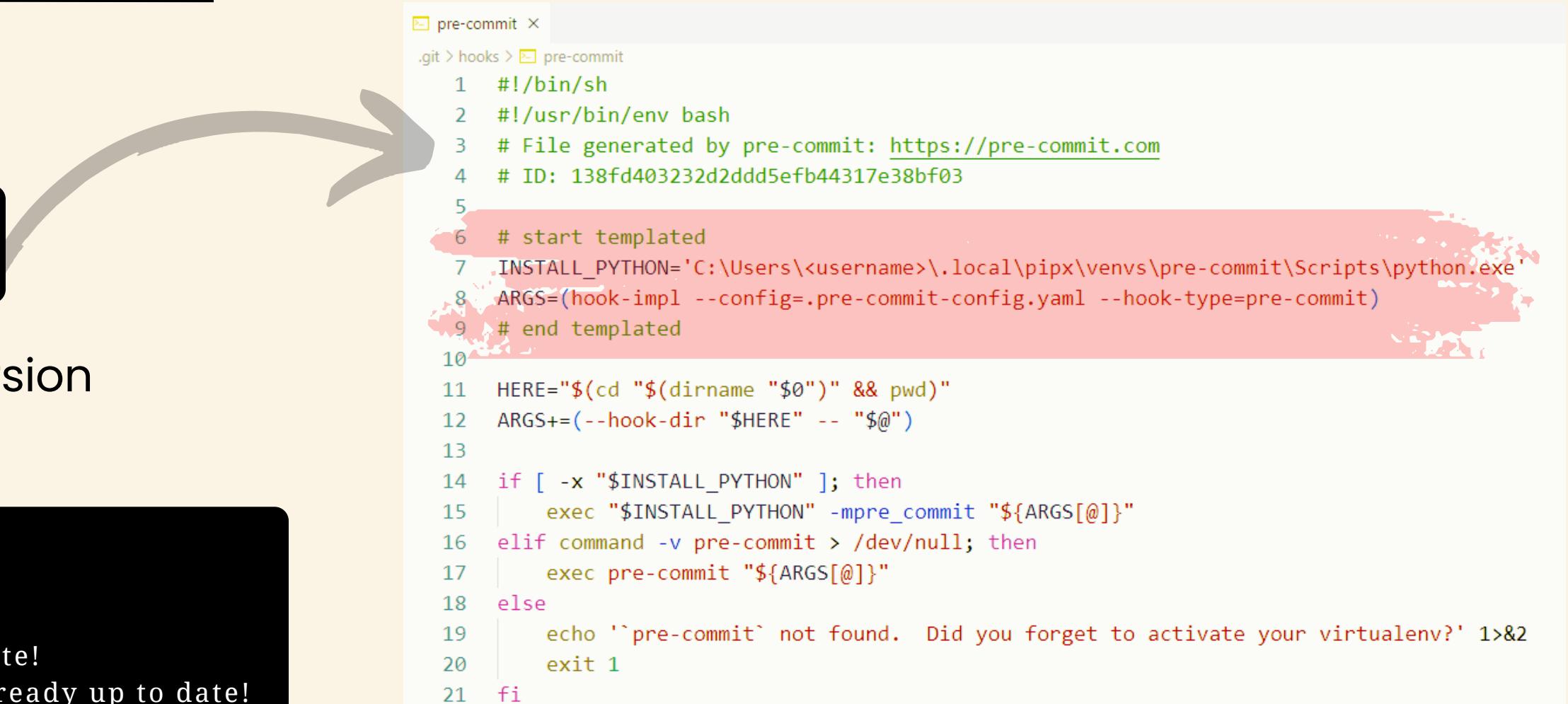
PRE-COMMIT-HOOKS

Install the git hook scripts

```
(.venv) > pre-commit install  
pre-commit installed at .git/hooks/pre-commit
```

To update your hooks to the latest version automatically

```
(.venv) > pre-commit autoupdate  
[https://github.com/psf/black] already up to date!  
[https://github.com/PyCQA/isort] already up to date!  
[https://github.com/pycqa/pydocstyle] already up to date!  
[https://github.com/charliermarsh/ruff-pre-commit] already up to date!  
[https://github.com/sqlfluff/sqlfluff] updating 2.1.0 -> 2.1.1  
[https://github.com/pre-commit/pre-commit-hooks] already up to date!
```



```
pre-commit x  
.git > hooks > pre-commit  
1  #!/bin/sh  
2  #!/usr/bin/env bash  
3  # File generated by pre-commit: https://pre-commit.com  
4  # ID: 138fd403232d2ddd5efb44317e38bf03  
5  
6  # start templated  
7  INSTALL_PYTHON='C:\Users\<username>\.local\pipx\venvs\pre-commit\Scripts\python.exe'  
8  ARGS=(hook-impl --config=.pre-commit-config.yaml --hook-type=pre-commit)  
9  # end templated  
10  
11 HERE=$(cd "$(dirname "$0")" && pwd)  
12 ARGS+=(--hook-dir "$HERE" -- "$@")  
13  
14 if [ -x "$INSTALL_PYTHON" ]; then  
15   exec "$INSTALL_PYTHON" -m pre_commit "${ARGS[@]}"  
16 elif command -v pre-commit > /dev/null; then  
17   exec pre-commit "${ARGS[@]}"  
18 else  
19   echo '`pre-commit` not found. Did you forget to activate your virtualenv?' 1>&2  
20   exit 1  
21 fi
```

This will bring the hooks to the latest tag on the default branch

PRE-COMMIT-HOOKS

Now **pre-commit** will run automatically on **git commit**

Hook idy

```
(.venv) > git commit -m "Update hooks and deps"
black.....(no files to check) Skipped
black-jupyter.....(no files to check) Skipped
isort.....(no files to check) Skipped
pydocstyle.....(no files to check) Skipped
ruff.....(no files to check) Skipped
sqlfluff-fix.....(no files to check) Skipped
sqlfluff-lint.....(no files to check) Skipped
check python ast.....(no files to check) Skipped
fix end of files..... Failed
- hook id: end-of-file-fixer
- exit code: 1
- files were modified by this hook

Fixing .pre-commit-config.yaml

trim trailing whitespace..... Passed
check yaml..... Passed
check toml.....(no files to check) Skipped
check json.....(no files to check) Skipped
```

Breathe

how?

PIP-TOOLS

JAZZ
BAND

REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS



COMPLIANCE

how?

GNU MAKE



CONSISTENCY

GNU MAKE

"GNU Make is a tool which enables the end user to build and install your package without knowing the details of how that is done. Make gets its knowledge from a file called the **makefile**.

Make is not limited to building a package. You can also use Make for or **anything else you want to do often enough** to make it worth while writing down how to do it."



> choco install make

MAKEFILE

```
1 .EXPORT_ALL_VARIABLES:
2 .PHONY: venv install update upgrade pre-commit check clean
3
4 GLOBAL_PYTHON = $(shell py -3.9 -c 'import sys; print(sys.executable)')
5 LOCAL_PYTHON = .venv\Scripts\python.exe
6 LOCAL_PIP_COMPILE = .venv\Scripts\pip-compile.exe
7 LOCAL_PIP_SYNC = .venv\Scripts\pip-sync.exe
8
9 setup: venv install pre-commit
10
11 ## Create an empty environment
12 venv: ${GLOBAL_PYTHON}
13     @echo "Creating .venv..."
14     - deactivate
15     ${GLOBAL_PYTHON} -m venv .venv
16
17 ## Install dependencies
18 install: ${LOCAL_PYTHON}
19     @echo "Installing dependencies..."
20     ${LOCAL_PYTHON} -m pip install --upgrade pip
21     ${LOCAL_PYTHON} -m pip install pip-tools
22     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements.txt --resolver=backtrack
23     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements-dev.txt --resolver=backtrack
24     ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"
25
26 ## Update dependencies
27 update: ${LOCAL_PYTHON} ${LOCAL_PIP_SYNC}
28     @echo "Updating dependencies..."
29     ${LOCAL_PYTHON} -m pip install --upgrade pip
```

MAKE RULES

"A **rule** in **makefile** tells Make how to execute a series of **commands** ... It also specifies a list of **dependencies** ... This list should include all files which are used as inputs to the commands in the rule



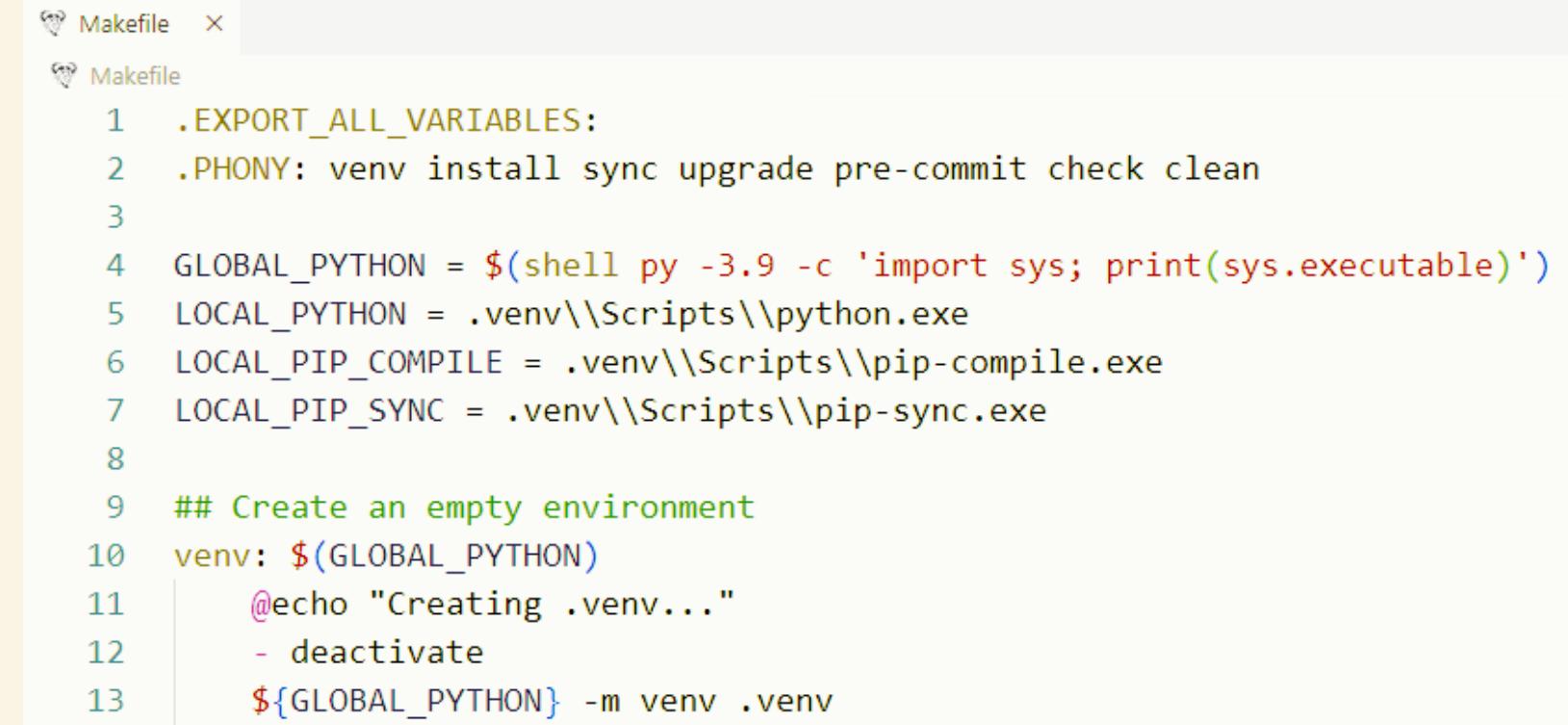
MAKFILE

```
Makefile M ×
Makefile
1 .EXPORT_ALL_VARIABLES:
2 .PHONY: venv install update upgrade pre-commit check clean
3
4 GLOBAL_PYTHON = $(shell py -3.9 -c 'import sys; print(sys.executable)')
5 LOCAL_PYTHON = .venv\Scripts\python.exe
6 LOCAL_PIP_COMPILE = .venv\Scripts\pip-compile.exe
7 LOCAL_PIP_SYNC = .venv\Scripts\pip-sync.exe
8
9 ## Create an empty environment
10 venv: ${GLOBAL_PYTHON}
11     @echo "Creating .venv..."
12     - deactivate
13     ${GLOBAL_PYTHON} -m venv .venv
14
15 ## Install dependencies
16 install: ${LOCAL_PYTHON}
17     @echo "Installing dependencies..."
18     ${LOCAL_PYTHON} -m pip install --upgrade pip
19     ${LOCAL_PYTHON} -m pip install pip-tools
20     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements.txt --resolver=backtr
21     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements-dev.txt --resolver=ba
22     ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"
23
```

MAKE RULES

Create a virtual environment

```
make venv  
"Creating .venv..."  
deactivate
```



A screenshot of a code editor showing a Makefile. The file contains the following content:

```
Makefile  
Makefile  
1 .EXPORT_ALL_VARIABLES:  
2 .PHONY: venv install sync upgrade pre-commit check clean  
3  
4 GLOBAL_PYTHON = $(shell py -3.9 -c 'import sys; print(sys.executable)')  
5 LOCAL_PYTHON = .venv\Scripts\python.exe  
6 LOCAL_PIP_COMPILE = .venv\Scripts\pip-compile.exe  
7 LOCAL_PIP_SYNC = .venv\Scripts\pip-sync.exe  
8  
9 ## Create an empty environment  
10 venv: $(GLOBAL_PYTHON)  
11     @echo "Creating .venv..."  
12     - deactivate  
13     ${GLOBAL_PYTHON} -m venv .venv
```

MAKE RULES

Install dependencies

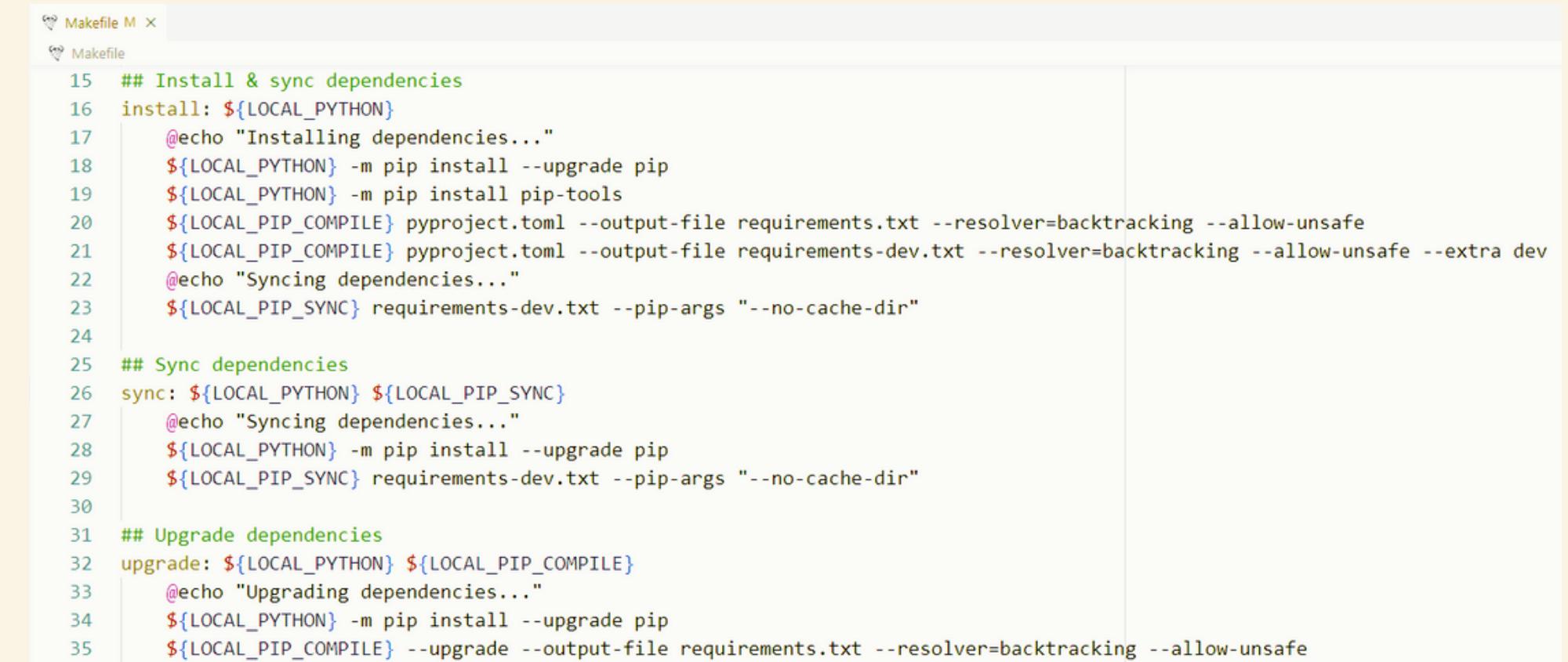
```
make install
"Installing dependencies..."
"Syncing dependencies..."
```

Update dependencies

```
make sync
"Syncing dependencies..."
```

Upgrade dependencies

```
make upgrade
"Upgrading dependencies..."
```



```
Makefile M ×
Makefile
15 ## Install & sync dependencies
16 install: ${LOCAL_PYTHON}
17     @echo "Installing dependencies..."
18     ${LOCAL_PYTHON} -m pip install --upgrade pip
19     ${LOCAL_PYTHON} -m pip install pip-tools
20     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements.txt --resolver=backtracking --allow-unsafe
21     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements-dev.txt --resolver=backtracking --allow-unsafe --extra dev
22     @echo "Syncing dependencies..."
23     ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"
24
25 ## Sync dependencies
26 sync: ${LOCAL_PYTHON} ${LOCAL_PIP_SYNC}
27     @echo "Syncing dependencies..."
28     ${LOCAL_PYTHON} -m pip install --upgrade pip
29     ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"
30
31 ## Upgrade dependencies
32 upgrade: ${LOCAL_PYTHON} ${LOCAL_PIP_COMPILE}
33     @echo "Upgrading dependencies..."
34     ${LOCAL_PYTHON} -m pip install --upgrade pip
35     ${LOCAL_PIP_COMPILE} --upgrade --output-file requirements.txt --resolver=backtracking --allow-unsafe
```

MAKE RULES

Install pre-commit hooks

```
make pre-commit  
"Setting up pre-commit..."
```

Setup

```
make setup  
"Creating .venv..."  
deactivate  
"Installing dependencies..."  
"Setting up pre-commit..."
```

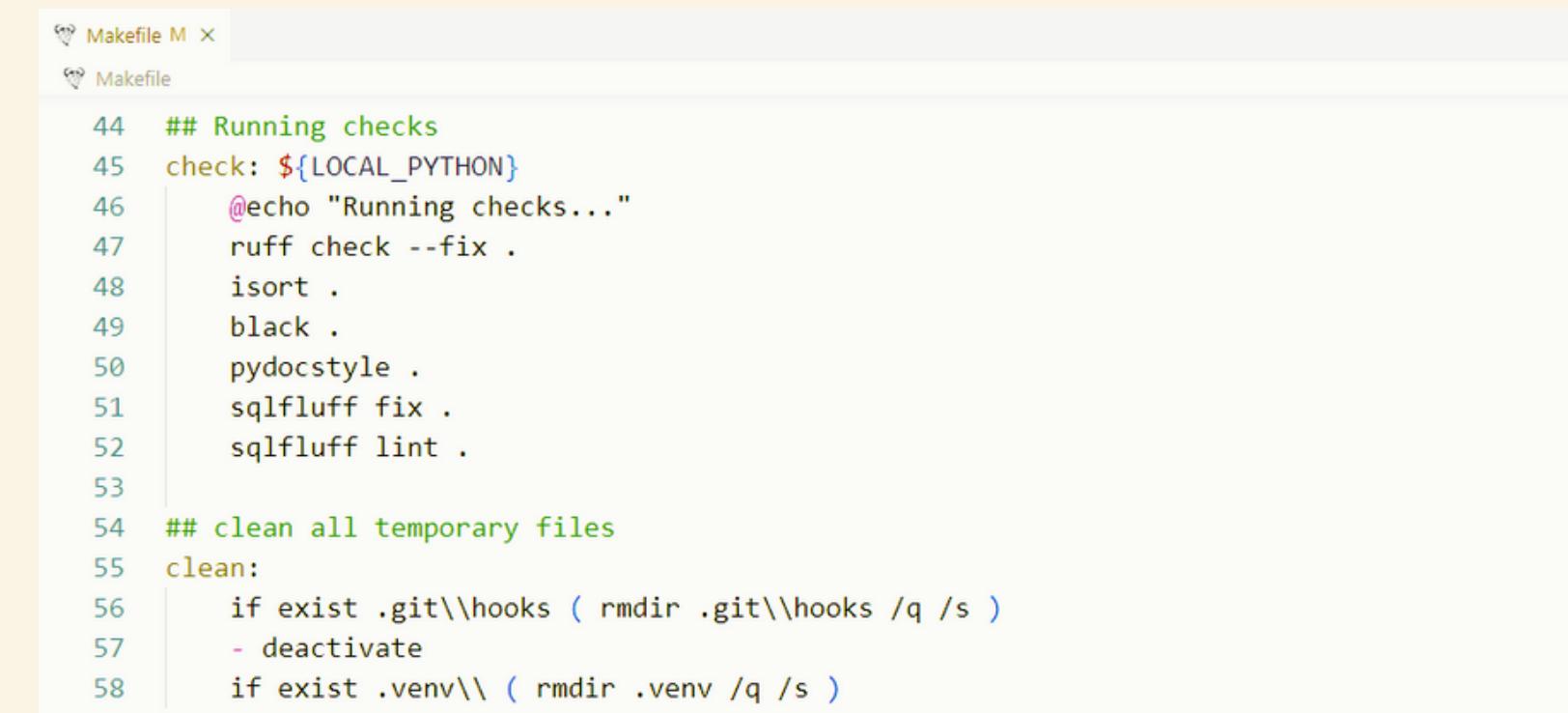


```
Makefile M X  
Makefile  
36 ## Install pre-commit hooks  
37 pre-commit:  
38     @echo "Setting up pre-commit..."  
39     pre-commit install  
40     pre-commit autoupdate  
41  
42 setup: venv install pre-commit
```

MAKE RULES

Run helper packages

```
make check
"Running checks..."
ruff check --fix .
isort .
Skipped 2 files
black .
All done! ☆☆
7 files left unchanged.
pydocstyle .
sqlfluff fix .
==== finding fixable violations ====
==== no fixable linting violations found ====
All Finished !
sqlfluff lint .
All Finished !
```



```
Makefile M ×
Makefile

44 ## Running checks
45 check: ${LOCAL PYTHON}
46     @echo "Running checks..."
47     ruff check --fix .
48     isort .
49     black .
50     pydocstyle .
51     sqlfluff fix .
52     sqlfluff lint .

53 ## clean all temporary files
54 clean:
55     if exist .git\hooks ( rmdir .git\hooks /q /s )
56     - deactivate
57     if exist .venv\ ( rmdir .venv /q /s )
```

Clean

```
make clean
```

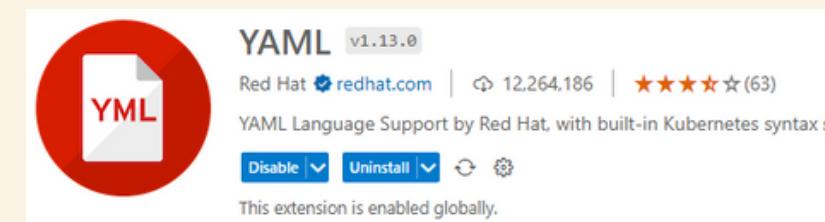
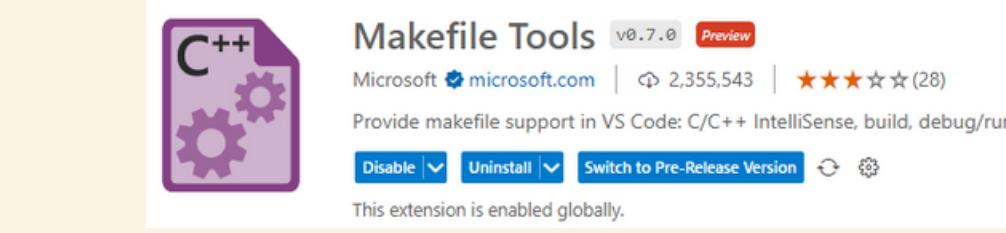
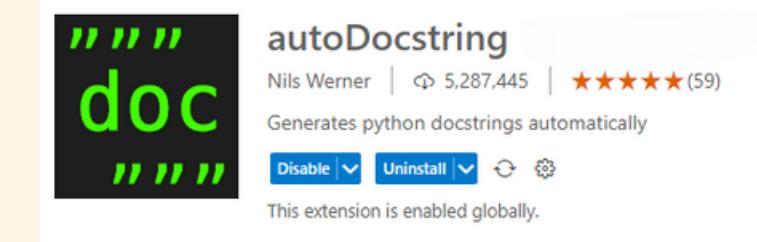
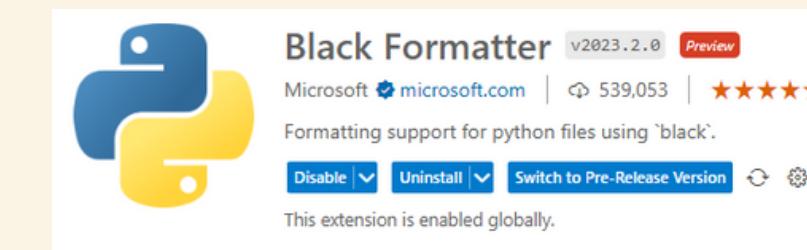
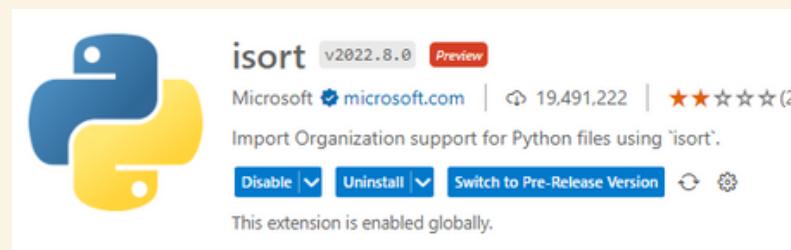
Breathe

BONUS

VSCODE INTEGRATION

VSCODE INTEGRATION

EXTENSIONS

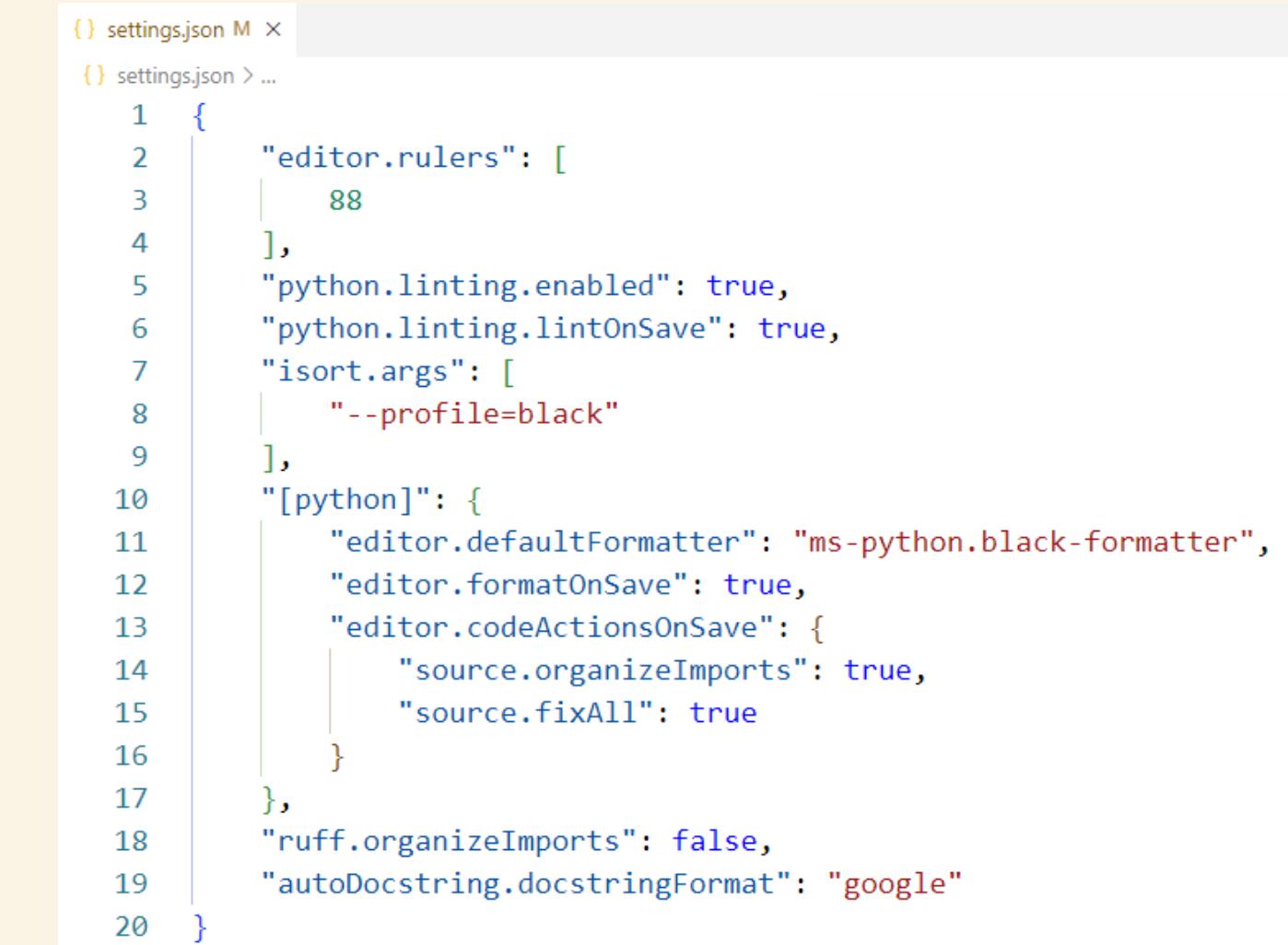


VSCODE INTEGRATION

SETTINGS

CTRL + SHIFT + P

Preferences: Open User Settings (JSON)



```
{} settings.json M x
{} settings.json > ...
1 {
2   "editor.rulers": [
3     88
4   ],
5   "python.linting.enabled": true,
6   "python.linting.lintOnSave": true,
7   "isort.args": [
8     "--profile=black"
9   ],
10  "[python]": {
11    "editor.defaultFormatter": "ms-python.black-formatter",
12    "editor.formatOnSave": true,
13    "editor.codeActionsOnSave": {
14      "source.organizeImports": true,
15      "source.fixAll": true
16    }
17  },
18  "ruff.organizeImports": false,
19  "autoDocstring.docstringFormat": "google"
20 }
```

Thank you!