

**Hook' il est beau notre code !**

---

# Python Rennes

---

Luc Sorel-Giffo ( @lucsorel )

Jérôme Marchand ( @Neken13 )



Figure 1. Hook'il est beau notre code !  
Lundi 7 novembre 2022

<https://www.meetup.com/fr-FR/python-rennes/events/289052290/>

# Quoi de n'œuf, Python ?



Figure 2. source : Heiko Kiera, Shutterstock - <https://www.aboutanimals.com/reptile/>

Lundi 7 novembre 2022 (Python Rennes)



# Python 3.11 est sorti !



Figure 3. <https://www.python.org/downloads/release/python-3110/>

Plus de détails :

- Les nouveautés de Python 3.11 - docstring
- Cool New Features for You to Try - RealPython

# Python 3.11 - Quelques nouveautés importantes

---

- interpréteur 10-60% + rapide
- temps de démarrage 10% + rapide
- → `Self:` pour des API fluent
- + de généricité dans les typages
- exceptions
  - lancement groupé d'exception
  - `ve.add_note('...')`
  - stack-trace plus explicite
- `asyncio`: context manager avec un groupe de tâches
- `@dataclass_transform` pour unifier les approches `@dataclass-like`
- `regexp` inclut le *groupage atomique* et les *quantificateurs possessifs* 😊



# Sortie de Poetry 1.2.0

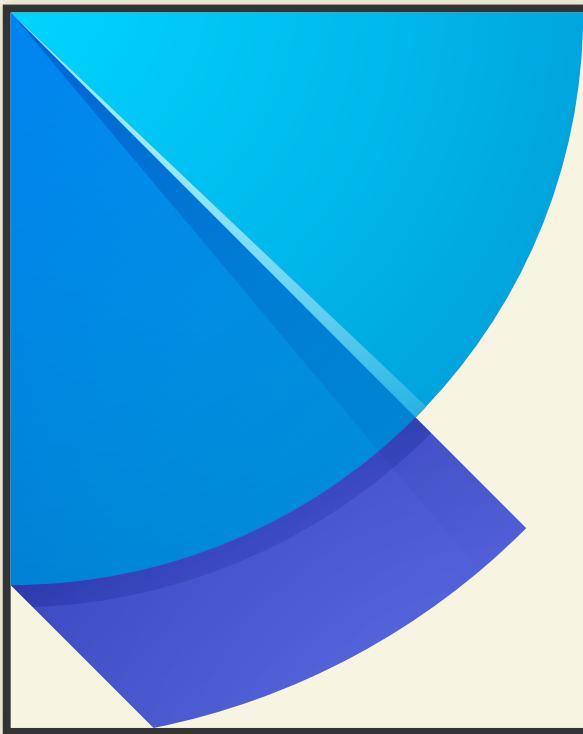


Figure 4. <https://python-poetry.org/blog/announcing-poetry-1.2.0/>

# Poetry 1.2.0 - Nouveautés cassantes

- nouvel installateur

```
curl -sSL https://raw.githubusercontent.com/python-poetry/poetry/master/get-poetry.py \
| python3 - --uninstall

curl -sSL https://install.python-poetry.org | python3 -
```

- abandon du support des projets 2.7
- utilisation exclusive de sha256 (abandon des hashes md5)
  - si pypiserver privé : voir redémarrer un pypiserver et regénérer des hashes sha256
  - si nexus privé : voir Nexus - PyPI repos should provide SHA256 hashes in /simple web interface

# Poetry 1.2.0 - Groupes de dépendances

Déclaration des groupes dans `pyproject.toml` :

```
# legacy [tool.poetry.dev-dependencies] -> [tool.poetry.group.dev.dependencies]
[tool.poetry.group.test.dependencies]
pytest = "^7.1.0"
pytest-cov = "^4.0.0"

[tool.poetry.group.profiling.dependencies]
optional = true
pyinstrument = "^4.4.0"
memray = "^1.4.0"
```

Utilisation des groupes :

```
# ajout d'une dépendance dans un groupe spécifié
poetry add pytest-mock --group test

# installation de dépendances optionnelles
poetry install --with profiling,docs

# installation sans dépendances obligatoires
poetry install --without test
```

## Poetry 1.2.0 - Autres nouveautés

- système de plugins
- installation sans mettre à jour les dépendances transitives

```
poetry install --sync
```

- utilisation du binaire python local pour l'environnement virtuel

```
poetry config --local virtualenvs.prefer-active-python true
pyenv local 3.11.0
poetry install
```

## Vérifications de dépendances - deptry

---

<https://fpgmaas.github.io/deptry/>

```
poetry add --group dev deptry  
deptry .
```

## **Tendances de popularité des bibliothèques - piptrends**

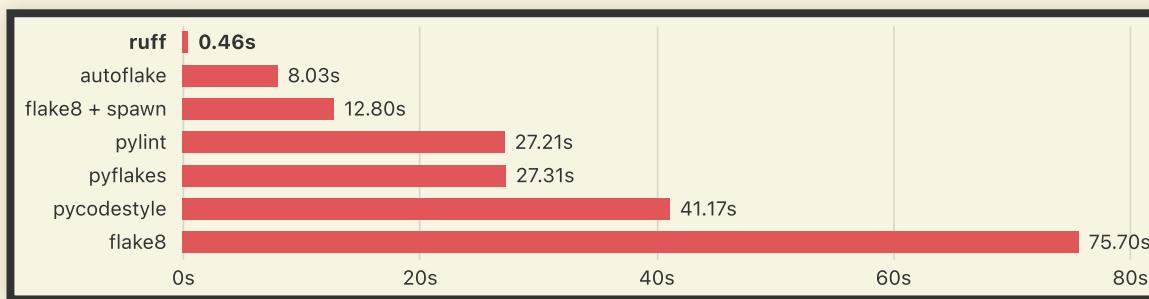
---

<https://piptrends.com/compare/pre-commit-vs-yapf-vs-black-vs-pylint-vs-ruff>

## Analyse de qualité de code

---

- refurb (nécessite Python 3.10+) :  
<https://github.com/dosisod/refurb>
- ruff (un linter Python ⚡-rapide écrit en rust) :  
<https://pypi.org/project/ruff/>



## Cette veille est collaborative

---

D'autres nouvelles ?

Merci pour vos contributions 😊

## **Qualité de code - du commit à la prod'**

---



*Quelque part dans un openspace  
de la startup nation...*

## **Qualité de code, conventions et relectures**

---

Pourquoi nous relisons-nous ?

## **Qualité de code, conventions et relectures**

---

Pourquoi nous relisons-nous ?

- le code doit rendre le service métier attendu

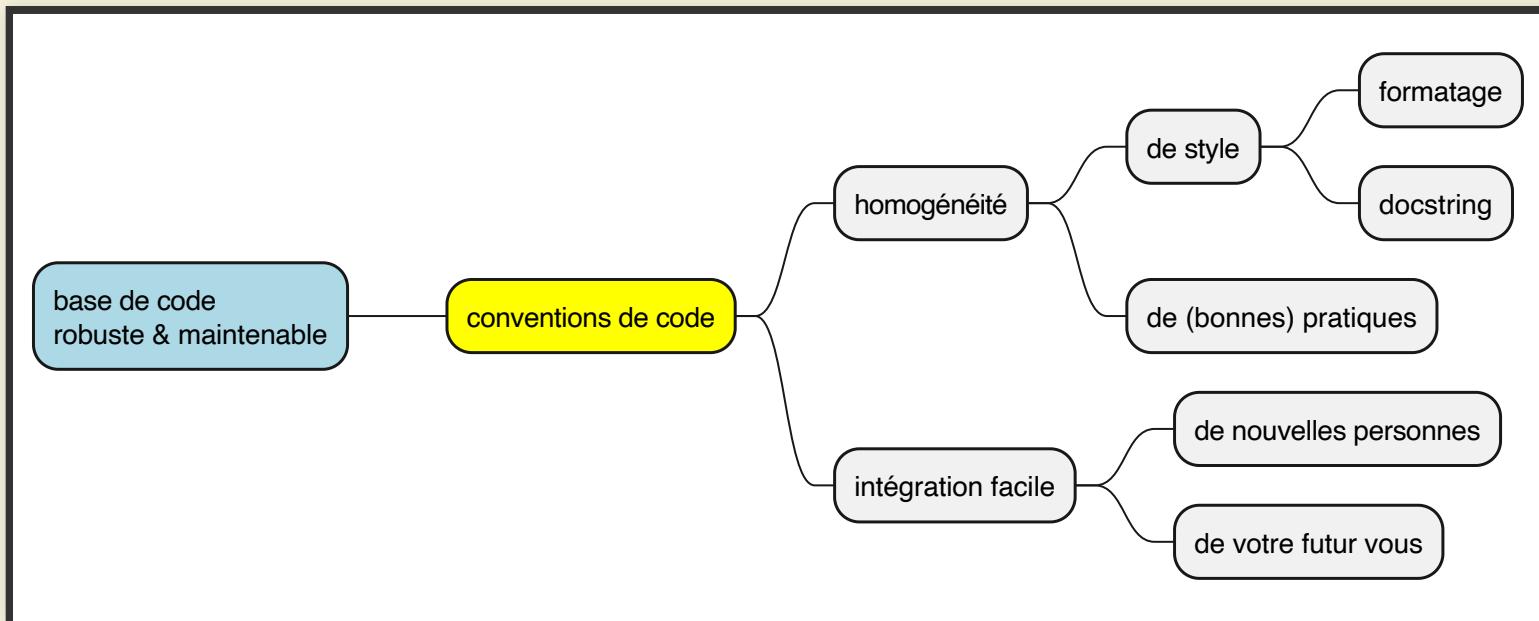
## Qualité de code, conventions et relectures

---

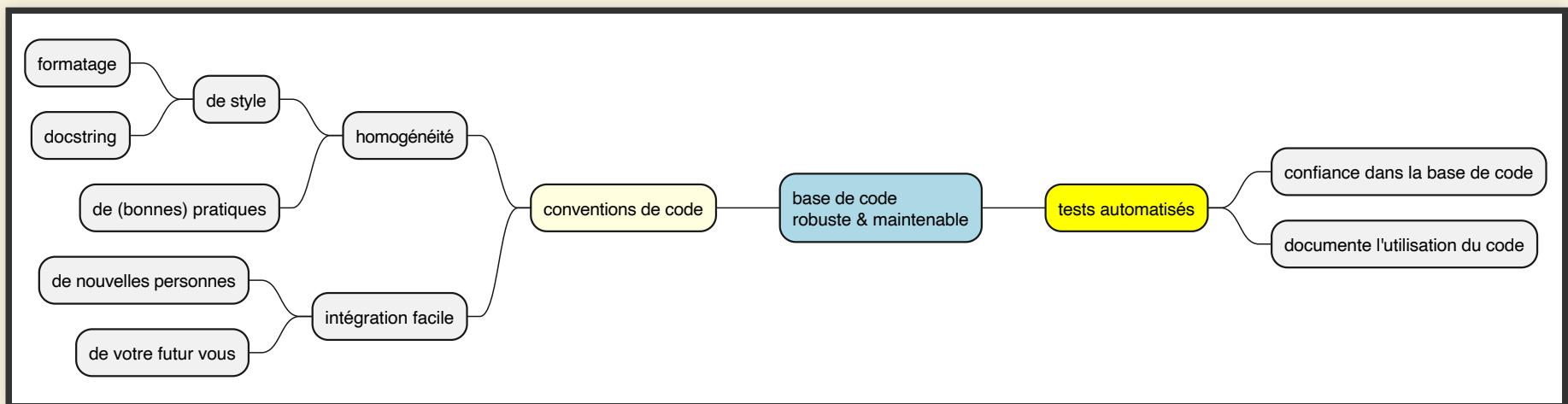
Pourquoi nous relisons-nous ?

- le code doit rendre le service métier attendu
- perpétuer la robustesse et la maintenabilité du service numérique

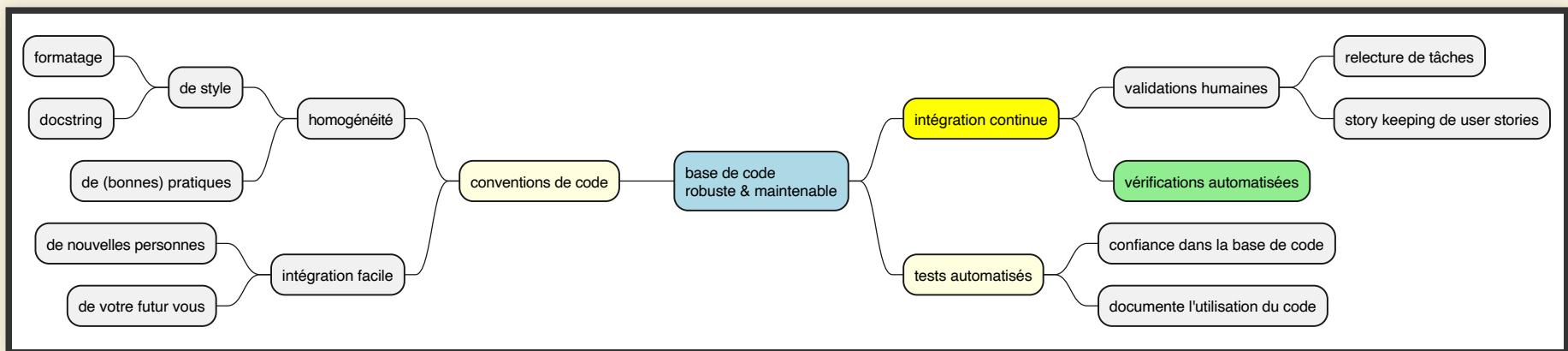
# Vers une base de code robuste et maintenable 1/3



## Vers une base de code robuste et maintenable 2/3



# Vers une base de code robuste et maintenable 3/3



## Vérifications automatisées

---

Que connaissez-vous comme vérifications automatisées de la qualité de code ?

## Vérifications automatisées

---

Que connaissez-vous comme vérifications automatisées de la qualité de code ?

- tests automatisés
  - en contrôlant la couverture de code : `pytest-cov`
  - en contrôlant la conso mémoire : `memray`

# Vérifications automatisées

---

Que connaissez-vous comme vérifications automatisées de la qualité de code ?

- tests automatisés
  - en contrôlant la couverture de code : `pytest-cov`
  - en contrôlant la conso mémoire : `memray`
- analyse statique de code
  - respect de formatage : `isort`, `black`, `yapf`
  - détection d'antipatterns : `pylint`, `flake8`, `ruff`, `perfline`

# **Et si l'analyse statique commençait au commit ?**

---

*(juste avant, en fait...)*

## Les hooks git

---

Conditionne une action git "classique" à la bonne exécution d'un script.

```
super-projet
  └── .git
    └── hooks/
      ├── commit-msg.sample
      ├── pre-commit.sample ••
      ├── pre-push.sample
      ├── pre-receive.sample
      └── ...
```

- retirer `.sample` pour activer le hook
- le contexte d'exécution est la racine du projet git, pas `.git/hooks`

# Example de hook de pre-commit

---

```
# 📄 .git/hooks/pre-commit
#!/bin/sh
echo "Pre-commit hook launched in $(pwd)"

# simulates an error code at exit
exit 1
```

```
mkdir super-projet
cd super-projet
git init -b main
git config ...

git touch test
git add test
git status
git commit -m "🎉"
git status
```

## Comment mutualiser les hooks ?

---

## Comment mutualiser les hooks ?

---

- créer un dossier contenant les hooks (`.git_hooks`, par exemple)
  - 🤢 faire un lien symbolique entre `.git_hooks` et `.git/hooks`
  - 👍 OU configurer git pour y chercher les hooks :  
`git config core.hooksPath ./git_hooks`

## Comment mutualiser les hooks ?

---

- créer un dossier contenant les hooks (`.git_hooks`, par exemple)
  - 🤢 faire un lien symbolique entre `.git_hooks` et `.git/hooks`
  - 👍 OU configurer git pour y chercher les hooks :  
`git config core.hooksPath ./ .git_hooks`
- 👍👍 utiliser un outil qui va gérer la *tambouille* entre les hooks git et les outils de vérification : `pre-commit`

# pre-commit

---

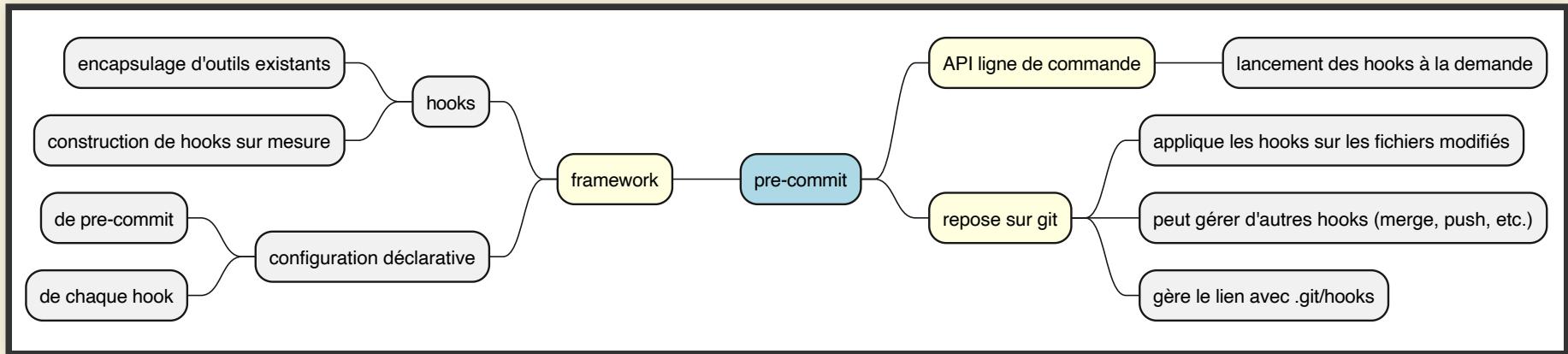


*A framework for managing and maintaining multi-language pre-commit hooks.*

- open-source (MIT license)
- 9.1k ⭐
- 84 releases

Figure 5. <https://pre-commit.com/>

# Tambouille-as-code



## Bibliothèque écrite en Python

- bibliothèque de développement de votre projet Python
- ou exécutable via une commande docker
- configuration dans `.pre-commit-config.yaml`

# Les hooks natifs

---

```
# 📄 .pre-commit-config.yaml
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v3.2.0
  hooks:
    # verifies the syntax of toml files
    - id: check-toml
    # trims all whitespace from the end of each line
    - id: trailing-whitespace
    # ensures that all files end in a newline and only a newline
    - id: end-of-file-fixer
    # prevents large files from being committed (>100kb)
    - id: check-added-large-files
      args: [--maxkb=100]
    # replaces "double quotes" with 'single quotes' wherever "it's applicable"
    - id: double-quote-string-fixer
```

Voir <https://pre-commit.com/hooks.html>

# Commandes

---

```
# câblage avec git (génère le script .git/hooks/pre-commit)
pre-commit install

# lance tous les hooks sur tous les fichiers
pre-commit run --all-files

# lance un hook sur tous les fichiers
pre-commit run end-of-file-fixer --all-files

# met à jour les hooks natifs
pre-commit autoupdate
```

Voir <https://pre-commit.com/#usage>

## Démo sur un projet Python

---

Application progressive des hooks sur expylliarmus

## Tri des imports avec isort 1/2

---

Ajout du hook dans `.pre-commit-config.yaml`:

```
- repo: https://github.com/pre-commit/mirrors-isort
  rev: v5.10.1
  hooks:
    - id: isort
      # required for python projects configured with a pyproject.toml file
      additional_dependencies: [toml]
```

## Tri des imports avec isort 2/2

---

Configuration de isort dans `pyproject.toml` :

```
[tool.isort]
# le nombre de caractères max par ligne doit être spécifié ici aussi, et être compatible
line_length = 120
# Hanging Grid Grouped (mode 5)
# from third_party import (
#     lib1, lib2, lib3, lib4,
#     lib5, etc.
# )
multi_line_output = 5
balanced_wrapping = false
# TESTS->known_tests : crée une section spécifique d'imports concernant le contenu des d
sections = ["FUTURE", "STDLIB", "THIRDPARTY", "FIRSTPARTY", "LOCALFOLDER", "TESTS"]
known_tests = ["tests"]
```

## Formatage de code avec yapf 1/2

---

Ajout du hook dans `.pre-commit-config.yaml`:

```
- repo: https://github.com/pre-commit/mirrors-yapf
  rev: v0.32.0
  hooks:
    - id: yapf
      name: Yapf
      # required for python projects configured with a pyproject.toml file
      additional_dependencies: [toml]
```

## Formatage de code avec yapf 2/2

Configuration de yapf (dans `pyproject.toml`) où les parenthèses sont refermées à la ligne :

```
[tool.yapf]
based_on_style = "facebook"
# voir la section https://github.com/google/yapf#knobs
COALESCE_BRACKETS = false
COLUMN_LIMIT = 120
DEDENT_CLOSING_BRACKETS = true
INDENT_DICTIONARY_VALUE = false
EACH_DICT_ENTRY_ON_SEPARATE_LINE = true
FORCE_MULTILINE_DICT = true
JOIN_MULTIPLE_LINES = false
SPACES_AROUND_DEFAULT_OR_NAMED_ASSIGN = false
SPLIT_BEFORE_CLOSING_BRACKET = true
SPLIT_BEFORE_DICT_SET_GENERATOR = true
SPLIT_COMPLEX_COMPREHENSION = true
SPLIT_BEFORE_EXPRESSION_AFTER_OPENING_PAREN = true
SPLIT_BEFORE_FIRST_ARGUMENT = true
```

Alternatives : `black` (formateur volontairement sans configuration)

## Analyse de pratiques avec ruff 1/2

---

```
- repo: https://github.com/charliermarsh/ruff-pre-commit
  rev: v0.0.95
  hooks:
    - id: ruff
```

## Analyse de pratiques avec ruff 2/2

---

Configuration de ruff dans `pyproject.toml` :

```
[tool.ruff]
line-length = 120
select = ["E", "F"]
ignore = ["E501"]
[tool.ruff.per-file-ignores]
"__init__.py" = ["F401"]
"path/to/file.py" = ["F401"]
```

# Méthodologie

---

## Méthodologie

---

- discuter les règles de formatage en équipe (communautés de pratiques Python)

## Méthodologie

---

- discuter les règles de formatage en équipe (communautés de pratiques Python)
- nuancer les règles indispensables et celles de goût

## Méthodologie

---

- discuter les règles de formatage en équipe (communautés de pratiques Python)
- nuancer les règles indispensables et celles de goût
- adapter au besoin métier (tests unitaires de TLN → guillemets doubles)

## Méthodologie

---

- discuter les règles de formatage en équipe (communautés de pratiques Python)
- nuancer les règles indispensables et celles de goût
- adapter au besoin métier (tests unitaires de TLN → guillemets doubles)
- tester les règles sur la base de code puis discuter les différences obtenues en équipe

## Méthodologie

---

- discuter les règles de formatage en équipe (communautés de pratiques Python)
- nuancer les règles indispensables et celles de goût
- adapter au besoin métier (tests unitaires de TLN → guillemets doubles)
- tester les règles sur la base de code puis discuter les différences obtenues en équipe
- accepter les compromis faits par le formateur : il y aura des cas où le résultat ne sera pas foufou, le but est de tendre vers une homogénéité de la base de code

## Autres cas d'application

---

Y a-t-il des (gens ayant des pratiques) devops dans la salle ?

## À quoi peut servir pre-commit ?

---

- trailing-whitespace
- end-of-file-fixer
- pretty-format-json
- check-added-large-files
- check-merge-conflict
- check-executables-have-shebangs
- check-case-conflict
- mixed-line-ending
- detect-aws-credentials
- detect-private-key
- check-json
- check-yaml
- ...

# Hook Custom

---

-  Projet Python
  -  `setup.cfg` pour décrire le package
  -  `setup.py` pour construire le package
  -  Un répertoire pour notre hook
    -  `__init__.py` pour avoir un package
    -  `main.py` le code de notre hook
- un repo git

Exemple de `check_json`

# Infrastructure as code

---

## pre-commit-terraform

```
repos:  
- repo: https://github.com/pre-commit/pre-commit-hooks  
  rev: v4.3.0  
  hooks:  
    ....  
- repo: https://github.com/antonbabenko/pre-commit-terraform  
  rev: v1.76.0  
  hooks:  
    - id: terraform_fmt  
    - id: terraform_docs  
      args:  
        - --args=--config=.terraform-docs.yaml  
    - id: terraform_tflint  
    - id: terraform_tfsec  
      args:  
        - >  
          --args=--format json  
          -e google-iam-no-project-level-service-account-impersonation,google-storage-en
```

## **Utilisation de pre-commit dans un mono-repo**

---

Dans le cas où pre-commit n'est utilisé que par les sous-projets Python

Rappels :

- pre-commit utilise la mécanique des hooks git
- le dossier de travail des hooks est donc la racine du mono-repo

## Script de compilation pour câbler les hooks git à pre-commit

---

- utiliser des Makefile`s avec une commande `make lint racine qui appelle les make lint de chaque sous-projet
- le make lint du projet Python lance pre-commit

```
lint:  
# vérifie qu'il y a des fichiers modifiés dans le sous-projet  
ifneq ($($shell git diff-index --quiet HEAD .; echo $$?), 0)  
    poetry run pre-commit run --all-files  
endif
```

# Une configuration pre-commit par sous-projet

---

```
files: ^a-python-subproject/  
  
repos:  
- repo: https://github.com/pre-commit/mirrors-yapf  
  rev: v0.32.0  
  hooks:  
  - id: yapf  
    name: Yapf  
    # this dependency is necessary if the python project is configured with a pyproj  
    additional_dependencies: [toml]  
    # goes in the sub-project and checks the format of the production and test codes  
    entry: sh -c "cd a-python-subproject \  
      && yapf --in-place -vv --recursive a_python_subproject tests"
```

## **Pre-commit en intégration continue**

---

## La commande du mal

---

```
git commit --no-verify -m "XD ahahaha !"
```

→ lancer les hooks dans l'intégration continue aussi

## Étape de lint dans l'intégration continue 1/2

---

Pre-commit repose sur git :

- installer git sur le serveur qui lance le job de lint
- initialisation minimale git du projet

```
apt-get install --no-install-recommends -y git
cd my-project
git init .
git add -A
poetry run pre-commit run --all-files
```

## Étape de lint dans l'intégration continue 2/2

---

Ou avec Docker

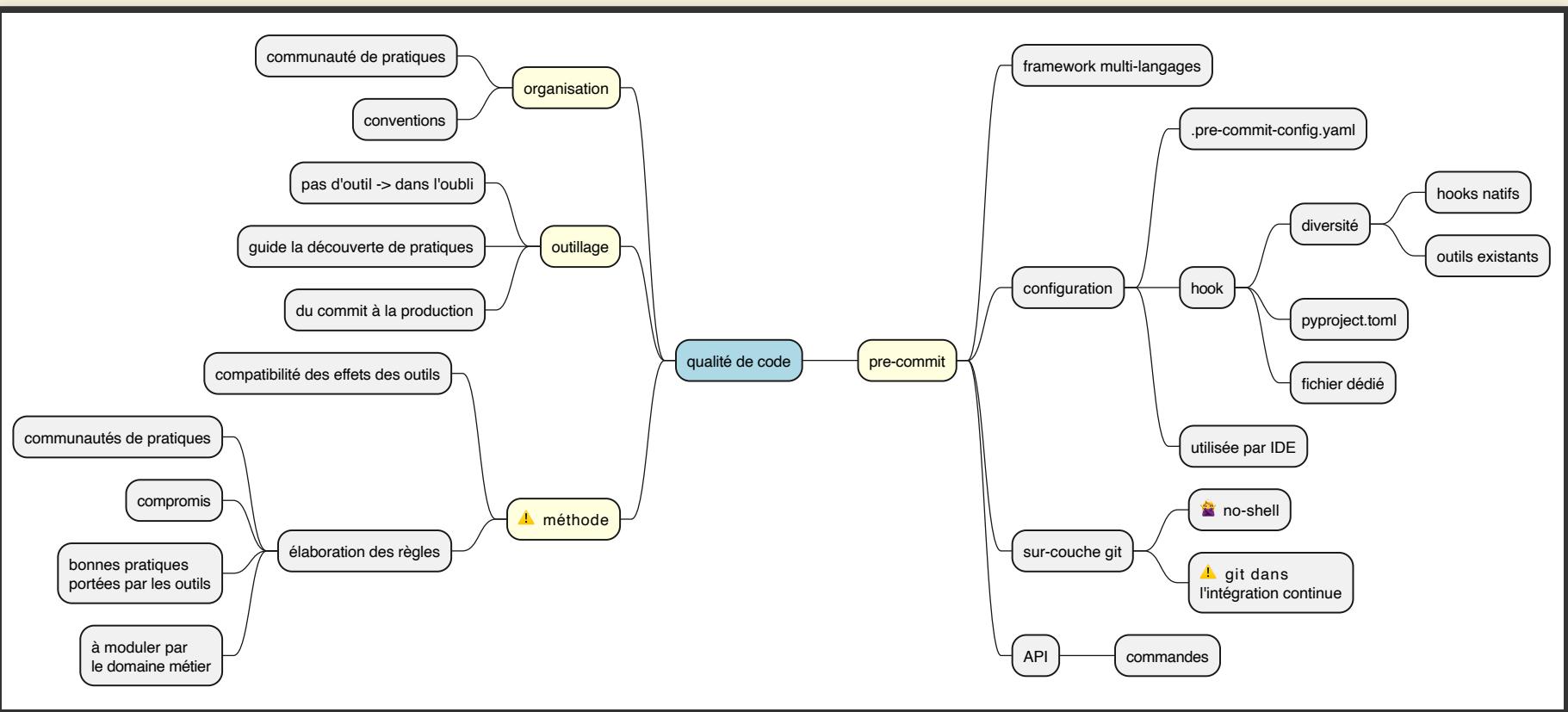
```
docker run --rm -v $(pwd):/data fxinnovation/pre-commit run -a
```

# Conclusions

---



À ramener dans votre pensine...



**Merci !**

---

Des questions ?

Des retours d'expérience ?