Bien tester son code en Python

Pytest, Fixtures, Mock et plus...

Yannick Stoll

Cemosis

TABLE OF CONTENTS

Pourquoi tester son code

- Plus rapide et robuste que les tests humains
- Permet de rapidement déceler les bogues
- Gain de confiance lors du développement
- Tests d'intégration et tests unitaires

Frameworks disponibles

- Unittest : inclut dans la bibliothèque standard de Python
- Pytest : doit être installé (min : Python 3.6), plus flexible, syntaxe simplifiée

Pytest

- Découverte automatique des tests
- Info détaillée sur les assertions qui ont échoué
- Peut-être utilisé en collaboration avec UnitTest
- Riche mécanisme de fixtures (paramétrisation des tests)
- Environnement de plugins associées

Premier test avec Pytest

Règles de découvertes

- `pytest Example_Tests_python/Presentation_de_Pytest/`
- Les noms des tests et des fichiers doivent commencer par "test_" ou se terminer par "_test"
- Pour les classes : Test comme dans TestExample (Pas de méthode __init__)
- `pytest --collect-only`, permet de voir quels tests pytest détecte

Options

- `pytest -k "failed and not suceed"`
  choix des tests
- `pytest -v`
  noms des tests explicites
- `pytest -s`
  les sorties ne sont pas capturées etc...

Remarques

- Les tests doivent être executés souvent (faire des tests rapides)
- Possibilité de tester qu'un code soulève une erreur
- Riche mécanisme d'introspection sur les asserts.
- Par rapport à UnitTest, assert* rendus inutiles

Organiser ses tests en classes

Remarques

- Pytest permet de regrouper des tests sous forme de classes
- Sous-classer les classes n'est pas nécessaire
- Les classes doivent commencer par "Test"
- `pytest -k TestDiv -v`

Avantages

- Organisation des tests
- Partager des fixtures au niveau de la classe
- Marquer les tests au niveau de la classe

Bonnes pratiques d'intégration

```
setup.py
mypkg/
    __init__.py
    app.py
    view.py
tests/
    test_app.py
    test_view.py
    ...
```

```
setup.py
mypkg/
    __init__.py
    app.py
    view.py
    test/
        __init__.py
        test_app.py
        test_view.py
        ...
```

Tests en dehors du paquet

- Permet de ne pas distribuer les tests en même temps que le paquet
- Les tests peuvent s'executer sur une version installée avec `pip install .` ou sur une copie locale `pip install --editable .`
- Si pas de fichier setup.py, on peut utiliser `python -m pytest` pour executer les tests directement contre la copie locale
- Attention cependant aux conflits de noms, mieux vaut avoir des noms de fichiers de tests uniques

Tests au sein du paquet

- Utile si on souhaite distribuer les tests avec l'application
- Les tests peuvent être lancés avec `pytest --pyargs mypkg`

pytest.ini

- Fichier d'initialisation de pytest
- Est prioritaire sur les autres fichiers de configurations/packaging (tox.ini, setup.cfg, pyproject.toml etc...)
- A placer à la racine du dépôt, là où est appelé pytest

Options

- minversion = : permet de préciser la version minimale de pytest
- addopts = : arguments par défauts
- testpaths = : où chercher les tests
- python_files = : permet de redéfinir le comportement par défaut etc..
- Voir `pytest -h` , pour une liste exhaustive.

```
# content of pytest.ini
[pytest]
minversion = 6.0
addopts = -s -v
testpaths =
     tests
     integration
python_files = check_*.py
python_classes = Check
python_functions = check_*
```

Mécanisme des fixtures

Présentation

- Permettent d'initialiser les tests
- Passage aux tests par arguments
- Définit par des décorateurs

Avantages

- Reproductibilité, indépendance
- Facilement maintenables
- Peuvent servir à marquer/paramétriser les tests
- Peuvent appeler d'autres fixtures

Fixtures disponibles dans pytest

- monkeypatch : modifie temporairement les classes, fonctions etc...
- tmpdir : permet de créer un répertoire temporaire pour le test
- capsys : capture, en tant que texte, les sorties de sys.stdout et sys.stderr
- voir documentations et plugins disponibles ...

Isoler ses tests

Principes

- Chaque test doit être indépendant des autres
- Après le test, l'application doit retrouver son état initial
- Bonne pratique : exécuter ses tests dans un ordre aléatoire

Implémentation dans pytest

- Utilisation d'expression générateur, expression yield
- Tout ce qui précède yield est executé avant le test
- Tout ce qui suit yield est executé après le test
- Sauf si le test soulève une exception !

```
@pytest.fixture()
def ma_fixture():
# execution avant le test
yield "toto"
# execution après le test
```

Tests paramétriques

Principes

- Les fixtures peuvent être paramétrées
- Les tests qui en dépendent seront appelés une fois par jeu de paramètres
- Meilleur exhaustivité dans les tests (tests avec plusieurs cas possibles)

Méthodes disponibles

- Soit par `@pytest.fixture()` (accès aux paramètres via la fixture request)
- Soit par `@pytest.mark.parametrize`, à appliquer directement aux tests

Mécanisme des mocks

Principes

- Utile si le code fait appel à des données globales (ENVVAR, sys.path) ou à un accès distant (elasticsearch, smtp etc...)
- Permet de modifier (imiter) les attributs de ces parties du code
- Permet un gain en rapidité
- Permet de faire tourner ses tests hors-ligne

Implémentation dans pytest : fixture monkeypatch

- monkeypatch.setattr(obj, name, value, raising=True) : permet de modifier un attribut (monkeypatch.delattr pour le supprimer)
- monkeypatch.setenv, monkeypatch.delenv permet de modifier la valeur des variables d'environnement
- monkeypatch.setitem, monkeypatch.delitem permet de modifier la valeur d'un dictionnaire global
- monkeypatch.chdir(path), permet de changer le répertoire courant pour le test

Les doctests : présentation

Principes

- Sert à introduire des tests unitaires dans les docstrings
- La docstring est parsée automatiquement par python
- Début du test marqué par >>>
- Fin des tests marqué par un saut de ligne

Avantages

- Permet de documenter sa docstring via des exemples
- Force à maintenir les docstrings à jour
- Executé soit via python ou via pytest : `pytest --doctest-modules`

Options # doctest : +FLAG

- SKIP : ignore la doctest
- ELLIPSIS : ignore la partie de la sortie marquée par `...`
- NORMALIZE_WHITESPACE : normalise les espaces dans la sortie
- `<BLANKLINE>` : permet de marquer une ligne vide dans la sortie (pas la fin des tests)

Les doctests

Limitations (voir exemples)

- La sortie doit exactement correspondre à ce que retourne l'interpréteur
- Attention aux sorties aléatoires (id(), repr())
- Attention aux dictionnaires, l'ordre de sortie n'est pas garantie!!
- Les caractères spéciaux doivent être échapés deux fois (i.e : `\\n`)

Quand utiliser les doctests

- Pour de petits tests sur des fonctions simples et indépendantes
- Généralement, on utilisera un mélange de doctests et de tests

Conclusion

- Pytest est un framework flexible pour les tests en python :
  - Découverte automatique des tests
  - Riche mécanisme d'introspection sur les assert
- Riche mécanisme de fixtures :
  - Injection de dépendances, tests reproductibles
  - Peuvent être imbriquées.
  - Permettent d'executer du code avant/après le test (isolation)
- Les mocks :
  - Permettent d'imiter une fonction faisant appel à une ressource extérieur
  - Permettent aussi de changer les variables globales durant les tests
  - Tous les mocks sont défaits à la fin du test

Conclusion

Aller plus loin

- Voir la classe MagickMock dans UnitTest en conjonction avec pytest
- Usine à fixtures (métaprogrammation)
- Pensez à regarder les plugins pour les besoins spécifiques

Références

- Doc officiel : `https://docs.pytest.org/`
- MOOC sur les tests python : `https://openclassrooms.com/fr/courses/4425126-testez-votre-projet-avec-python`
- Introspection des asserts : `http://pybites.blogspot.com/2011/07/behind-scenes-of-pytests-new-assertion.html`
- Doctests : `https://pymotw.com/2/doctest/`
- Mocks : `https://medium.com/@bfortuner/python-unit-testing-with-pytest-and-mock-197499c4623c`
- Tuto sur les tests : `https://semaphoreci.com/community/tutorials/testing-python-applications-with-pytest`
- Site à ne pas mettre entre toutes les mains, disponibilité du serveur douteuse!! `http://sametmax.com/un-gros-guide-bien-gras-sur-les-tests-unitaires-en-python-partie-1/`