Outils Python

Objectifs

— Savoir créer un environnement virtuel
— Savoir tester une fonction avec pytest
— Comprendre les couvertures de test avec coverage

Exercice 1 Le module venv permet de créer un environnement Python propre à un projet, avec ses propres modules installés, indépendamment de ceux installés sur le système.

Voici les étapes pour créer une environnement virtuel dans le dossier `~/nosave/py3` :

```
python3 -m venv $HOME/nosave/py3
$HOME/nosave/py3/bin/python --version            # python3, normalement !
```

Pour l'activer, il suffit alors de faire :

```
source $HOME/nosave/py3/bin/activate
which python                # Celui de py3 !
which pip                   # Aussi celui de py3
```

L'outil `pip` permet d'installer des paquets Python. Par exemple, pour installer l'outil de test pytest, on peut faire :

```
pip install pytest
```

Pour désactiver l'environnement virtuel Python, il suffit de faire :

```
deactivate
```

Activer l'environnement oblige à taper une commande un peu longue. On peut définir un alias pour le faire plus rapidement :

```
alias p3="source $HOME/nosave/py3/bin/activate"
```

Pour que cet alias soit permanent, on peut l'ajouter dans le fichier `$HOME/.bashrc`.

Exercice 2 : Pourquoi et comment tester

Le langage Python s'appuie sur un typage dynamique. Ceci signifie que quand un programme Python est chargé par l'interpréteur, aucune vérification du programme n'est faite à part le respect de la syntaxe Python. Il est donc d'autant plus important de tester son programme et de s'assurer que l'ensemble de ses instructions ont été exécutées au moins une fois. Ceci ne garantira pas que le programme est correct mais permettra de détecter des erreurs !

Considérons la fonction du fichier `minimum.py` qui calcule le minimum de ses deux paramètres. Elle contient manifestement une erreur que nous ne corrigerons pas maintenant.

Le fichier `test_minimum.py` est un programme de test pytest de la fonction minimum. pytest considère les fichiers dont le nom commence par `test_` (ou se termine par `_test`) comme des programmes de test et, dans ces fichiers, les fonctions dont le nom commence par `test` sont des fonctions de test. Les tests s'appuient sur l'instruction `assert` qui lève une exception si l'expression booléenne est fausse et ne fait rien sinon.

2.1. *Installation de pytest.* L'outil *pytest* ne fait pas partie de la livraison standard de Python. On peut l'installer grâce à l'installateur de paquets fourni avec Python : *pip*. On peut installer *pytest* en faisant [1] :

```
pip install pytest
```

2.2. *Exécution des tests.* Exécuter [2] le programme de test avec `pytest`. Par exemple :

```
pytest test_minimum.py
```

Est-ce que le test réussi ?

2.3. *Couverture des instructions.* Un outil de couverture de code permet de recenser les instructions qui ont été exécutées et celles qui ne l'ont pas été pendant une exécution donnée. C'est donc un outil qui nous donne des indications pour identifier de nouveaux tests à faire en s'appuyant sur la structure du programme. On parle de tests structurels ou tests en boîte blanche.

Si `coverage` n'est pas déjà installé, faire : `pip install coverage`

Pour exécuter [3] les tests sous *coverage*, on peut faire [4] :

```
coverage run -m pytest test_minimum.py
```

Les résultats sont conservées dans un fichier caché (.coverage sous Linux). La commande suivante [5] permet d'engendrer un rapport au format texte [6] :

```
coverage report -m minimum.py
```

Consulter les résultats.

2.4. *Couverture des enchaînements.* En plus des instructions, `coverage` peut aussi analyser les enchaînements. Par exemple, pour **`if`** (même sans **`else`**), il faudrait faire un test pour lequel la condition est vraie et l'autre où elle est fausse. Ainsi, on teste les deux enchaînements possibles après l'évaluation de la condition du **`if`**.

Pour demander aussi l'analyse des enchaînements, il faut ajouter l'option `--branch` :

1. On peut aussi faire : `python -m pip install pytest`. Cette version a pour avantage d'utiliser la version courante de python pour installer les nouveaux paquets.
2. On peut aussi exécuter `pytest` depuis l'interpréteur Python en faisant `import pytest` puis `pytest.main(['test_minimum.py'])`.
3. On peut aussi exécuter coverage depuis un programme Python : voir `https://stackoverflow.com/questions/61184932/code-coverage-of-a-file-using-coverage-py-using-a-python-script`
4. `-m` demande à Python d'exécuter le module dont le nom suit. `pytest` est le module qui correspond à `pytest`.
5. L'option `-m` affiche aussi les numéros des lignes manquantes.
6. On peut aussi engendrer un rapport au format HTML avec `coverage html minimum.py` qui engendre le dossier `htmlcov`. Il suffit alors d'ouvrir le fichier `htmlcov/index.html` avec un navigateur. On peut alors cliquer sur la ligne `minimum.py` pour consulter les résultats concernant ce module.

```
coverage run --branch -m pytest test_minimum.py
```

Bien sûr, il faut ré-engendrer les résultats :

```
coverage report -m minimum.py
```

Consulter les résultats et constater que la couverture du `if` est marquée partielle. Comment interpréter les numéros affichés à droite ?

2.5. *Exploiter les résultats de couverture.* Compléter les tests pour atteindre un taux de couverture de 100% des instructions et branchements. Que constate-t-on ? Que faire ?

2.6. *Complétude.* Est-ce qu'un taux de couverture de 100% des instructions et enchaînements avec des tests qui réussissent est suffisant pour conclure que le programme est correct ?