

d) Commandes Pytest et html report

Les commandes pytest

Appels de pytest

On exécutera pytest grâce à la commande suivante :

```
python -m pytest (ou tout simplement pytest)
```

Debug

Si l'on souhaite afficher les informations issues de print dans les fonctions de test, on pourra utiliser la commande suivante :

```
pytest -s
```

Arrêter l'exécution après N échecs

```
pytest -x          # stop after first failure  
pytest --maxfail=2 # stop after two failures
```

Exécuter le test sur un fichier ou une fonction

Si l'on souhaite :

- Exécuter le test sur un fichier :

```
pytest test_mod.py
```

- Exécuter le test sur un dossier :

```
pytest testing/
```

- Exécuter le test sur une fonction spécifique d'un fichier :

```
pytest test_mod.py::test_func //Test la fonction test_func du fichier  
test_mod.py  
pytest test_mod.py::TestClass::test_method //Test la méthode d'une classe
```

- Exécuter le test sur des markers :

```
pytest -m slow //Exécutera les tests décorés par @pytest.markerslow
```

Modification des retours

On peut modifier les retours des tests dans la console :

```
pytest -l          # show local variables (shortcut)

pytest --tb=auto    # (default) 'long' tracebacks for the first and last
                   # entry, but 'short' style for the other entries
pytest --tb=long    # exhaustive, informative traceback formatting
pytest --tb=short   # shorter traceback format
pytest --tb=line    # only one line per failure
pytest --tb=native  # Python standard library formatting
pytest --tb=no      # no traceback at all
```

Contenu du résumé

Par résumé, on entendra les dernières lignes du retour de test :

```
===== short test summary info =====
SKIPPED [1] test_example.py:22: skipping this test
XFAIL test_example.py::test_xfail
  reason: xfailing this test
XPASS test_example.py::test_xpass always xfail
ERROR test_example.py::test_error - assert 0
FAILED test_example.py::test_fail - assert 0
== 1 failed, 1 passed, 1 skipped, 1 xfailed, 1 xpassed, 1 error in 0.12s ==
```

L'argument -r acceptera toutes ces valeurs:

- f - failed
- E - error
- s - skipped
- x - xfailed
- X - xpassedp - passed
- p - passed
- P - passed with output

Les valeurs peuvent aussi être condensées :

- "a" regroupe tout sauf "p" et "P"
- "A" regroupe tout

Question : À partir de quelle ligne de commande a été exécutée l'exemple précédent :

[Réponse](#)

L'exemple précédent a été exécuté grâce à la commande `pytest -ra`

Rediriger les erreurs dans des logs

On pourra rediriger les retours dans des logs grâce à la ligne suivante :

```
pytest --resultlog=path //path sera le nom du fichier dans lequel les logs  
seront rediriges
```

Exemple de contenu:

```
F example.py::test_add  
def test_add():  
>         assert add(2, 3) == 5  
E         assert -1 == 5  
E         +   where -1 = add(2, 3)  
  
example.py:6: AssertionError
```

HTML Reporting

Tout d'abord, installons la librairie qui nous permettra d'avoir des rapports de test plus simples à lire.

```
pip install pytest-html
```

Une fois que cette librairie est installée, la commande pytest sera enrichie d'arguments.
Tapons tout simplement:

```
pytest --html=report.html
```

Et allons voir ensemble la page qui s'est créée à l'endroit où nous avons lancé la commande :

report.html

Report generated on 19-Feb-2021 at 15:24:53 by pytest-html v1.22.1

Environment

Packages	Py: '1.10.0', pytest: '6.0.1', pluggy: '0.13.1'
Platform	Darwin-19.0.0-x86_64-i386-64bit
Plugins	0/metadata: u'1.11.0', u/html: u'1.22.1', u/ov: u'2.11.1'
Python	2.7.16

Summary

1 tests ran in 0.11 seconds.

(Un)check the boxes to filter the results.

0 passed, 0 skipped, 1 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration	Links
Failed (hide details)	test_app.py::test_http_return	0.00	

```
tmpdir = local('/private/var/folders/50/f7j6kcd5dd_37g6c1nptmtw0000gn/T/pytest-of-keryiclain/pytest-20/test_http_return0'), monkeypatch = <pytest.monkeypatch.MonkeyPatch object at 0x10b7435b>  
  
def test_http_return(tmpdir, monkeypatch):  
    p = tmpdir.mkdirc("program").join("test.txt")  
    # run script  
    app.main(["--dest", str(p)])  
    print(p)  
    local_res = "Hello"  
    with open(str(p), "r") as f:  
        text = f.readlines()[0].strip()  
    > assert local_res == text  
E AssertionError: assert 'Hello' == 'Hello world'  
E  
E + Hello world  
  
test_app.py:14: AssertionError  
===== Captured stdout call =====  
/private/var/folders/50/f7j6kcd5dd_37g6c1nptmtw0000gn/T/pytest-of-keryiclain/pytest-20/test_http_return0/program/test.txt
```

C'est bien plus compréhensible que de lire dans une console, non ?!

Json reporting

Si l'on souhaite créer notre propre interface, on pourra créer des reports sous forme de json.
Pour cela, on installera la librairie à partir de la commande suivante:

```
pip install pytest-json-report
```

Et on utilisera les arguments de la librairie json :

```
pytest --json-report-file=json_report.json --json-report-indent=2 --json-report-summary
```

Définition des arguments:

- `--json-report` demande à ce que l'on crée un report en json
- `--json-report-file=PATH`, indique le nom du fichier json qui sera créé
- `--json-report-indent=2`, on met un peu en forme
- `--json-report-summary`, on n'affiche que le principal sinon c'est illisible !

Résultat :

```
{
  "environment": {
    "Python": "2.7.16",
    "Platform": "Darwin-19.6.0-x86_64-i386-64bit",
    "Packages": {
      "py": "1.10.0",
      "pytest": "4.6.11",
      "pluggy": "0.13.1"
    },
    "Plugins": {
      "json-report": "1.2.4",
      "metadata": "1.11.0",
      "html": "1.22.1",
      "cov": "2.11.1"
    }
  },
  "created": 1613745598.360417,
  "duration": 0.11444091796875,
  "exitcode": 1,
  "root": "/Users/kerylclain/Desktop/test",
  "summary": {
    "collected": 1,
    "total": 1,
    "error": 1
  }
}
```