doctest

Python 公式ドキュメント doctest --- 対話的な実行例をテストする

doctest は、テストコードをドキュメント(docstring)に埋め込むことができるテストフレームワークです。

テストモジュールやテスト関数等を別途容易しなくて良いので、 pytest, unittest と比べてテストコードの記述が簡単です。
「ドキュメントを書くついでにテストも書いておく」というくらいの感覚で使えます。

インストール

unittest は、Python に標準でバンドルされています。
なので、 `pip` コマンド等の実行なしですぐに使えます。

doctest が実行するテストの要件

doctest のテストランナーは、以下の要領でテストを実行します。

- 対話的 Python セッションのように見えるテキストを探し出す
- セッションの内容を実行する
- テキストに書かれている通りに振舞うかを調べる

doctest の書き方

doctest を含めた docstring の例を示します。

`doctests/doctest_ok.py`

```
def get_last_month(month):
    """ 指定された月の前月を取得する。(簡略番)

    :param month: 月を表す整数
    :return: 月を表す整数

    >>> get_last_month(1)
    12
    >>> get_last_month(2)
    1
    >>> get_last_month(7)
    6
    >>> get_last_month(13)
    Traceback (most recent call last):
        ...
    ValueError: 月は1-12の範囲で指定してください
    """
    if not isinstance(month, int):
        raise TypeError('月は整数で指定してください')
    if month < 1:
        raise ValueError('月は1-12の範囲で指定してください')
    if month > 12:
        raise ValueError('月は1-12の範囲で指定してください')

    if month == 1:
        return 12
    else:
        return month - 1


if __name__ == '__main__':
    import doctest

    doctest.testmod()
```

doctest の実行とレポートの出力

doctest によるテストの実行とテストレポート出力の方法をいくつか紹介します。

1. Python コマンドでファイルを実行する
2. Python シェルで doctest を実行する
3. doctest を unittest から呼び出す

1. Python コマンドでファイルを実行する

上記のように `if __name__ == '__main__':` 内で `doctest.testmod` メソッドを呼び出している場合は、 `python` コマンドを実行するだけで doctestを実行できます。

```
python month_funcs/doctest_ok.py
```

テストに成功した場合は何も出力されません。

エラーがあった場合は、以下のようにエラー内容が出力されます。

```
(venv) PS D:\project_dir> python month_funcs/doctest_ng.py
**********************************************************************
File "D:\project_dir\doctests\doctest_ng.py", line 11, in __main__.get_last_month
Failed example:
    get_last_month(7)
Expected:
    1
Got:
    6
**********************************************************************
1 items had failures:
   1 of   4 in __main__.get_last_month
***Test Failed*** 1 failures.
```

もっとも、以下のように `python` コマンドで末尾に `-v` オプションを指定すると、テスト実行過程を詳細に出力できます。
これにより、テストが成功した場合もその過程を把握できますし、少なくとも、「テストが実行された」ということは確認できます。

> `-v` は、英語の verbose (冗長な、詳細な) という単語の頭文字で、コマンドライン引数等でよく登場するオプションです。

```
python month_funcs/doctest_ok.py -v
```

```
(venv) PS D:\project_dir> python month_funcs/doctest_ok.py -v
Trying:
    get_last_month(1)
Expecting:
    12
ok
Trying:
    get_last_month(2)
Expecting:
    1
ok
Trying:
    get_last_month(13)
Expecting:
    Traceback (most recent call last):
        ...
    ValueError: 月は1-12の範囲で指定してください
ok
1 items had no tests:
    __main__
1 items passed all tests:
   3 tests in __main__.get_last_month
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
```

2. Python シェルで doctest を実行する

Python シェルから doctest を実行するには、以下のように `doctest.testmod` を呼び出します。

`doctest.testmod` の受け取る第一引数は、テスト対象のモジュールです。
(引数を省略すると、呼び出し元のモジュール `__main__` がテスト対象になります)

```
import doctest
from month_funcs import doctest_ng

doctest.testmod(doctest_ng)
```

そのほかの引数としては、 `verbose` くらいは覚えておくと良いかもしれません。
`verbose=True` を指定することで、詳細なテスト実行情報が表示されます。

```
import doctest
from month_funcs import doctest_ok

doctest.testmod(doctest_ok, verbose=True)
```

3. doctest を unittest から呼び出す

doctest は、unittest から呼び出すこともできます。
これにより、 doctest と unittest を組み合わせてテストを実行することも可能です。

また、複数のモジュールにある doctest を連続的に実行することもできます。

以下で `load_tests` 関数を定義しています。
これは、 unittest がテストモジュールを読みこんでテストスイートを作る際に呼び出される関数です。

この関数の中で、以下の要領で doctest を含むモジュールをテスト対象に追加します。
これにより、モジュール内の doctest も unittest 実行時のテストスイートに追加されます。

Python 公式ドキュメント doctest --- 対話的な実行例をテストする 単体テスト API

month_funcs/test_doctest_by_unittest.py

```
import doctest

from month_funcs import doctest_ok


def load_tests(loader, tests, ignore):
    """ロードされたテストスイートにドックストリングテストを追加する関数

    :param loader: テストローダー
    :param tests: テストスイート
    :param ignore: 無視する要素
    :return: 追加されたテストスイート
    """
    # ドックストリングテスト用のテストスイートを作成する
    test_suites = doctest.DocTestSuite(doctest_ok)

    # ロードされたテストスイートにドックストリングテストを追加する
    tests.addTests(test_suites)

    return tests
```

doctest のメリットとデメリット

doctest には、以下のような利点があります。

- ドキュメント作成時に手軽にテストコードを書けるので手軽
- ドキュメントとテストコードを一体化できる
- ドキュメントを見ただけで関数やメソッドの使い方が分かる

一方、以下のような欠点もあります。

- ドキュメントが肥大化してしまう
- テスト項目が多い場合は網羅的にテストが書かれているのか分かりにくくなる
- 後述の Coverage のようなツールを利用するには、 unittest との連携が必要になる

doctest を使うか pytest や unittest を使うかは、状況次第です。