プログラミング1

(第11回) ユニットテストの意義＆演習、モジュールの動作、タプル

1. プログラミング補足
    1. ユニットテスト演習
    2. モジュールの動作
2. Chapter 5.1 Tuples
    1. 変更できない、順序付きシーケンス集合
3. まとめ

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2019/prog1/

ユニットテストの補足

- ユニットテスト（doctest）の利用
  - テストの目的
  - 書き方・実行方法

何のためにユニットテストを書くのか？

- 関数が想定通りに動くことを「検証しやすくする」ため。
  - 毎回手動確認するのは辛い。
  - 「想定通り」の質が確認者に強く依存する。
  - 「動作例」を記述することで、想定している使い方を示すことにもなる。
- リファクタリング
  - 機能を保ったまま、コードを修正。

- テスト駆動開発
  - 背景: 開発現場でよくある状況
    - 「こういう入力が与えられた時に、こういう出力をする関数を書いて」
    - 入力と出力は分かっている。or 分からないなら、コードを書き始める前に明確にする。
  - テストを先に書くことで、その関数をどう動作させたいかを明確にする。
  - 書いたテストが通るようにコードを書く。

参考:
エンジニアのスキルを伸ばす「テスト駆動開発」を学んでみよう:
https://thinkit.co.jp/story/2014/07/30/5097

ユニットテスト演習

- https://github.com/naltoma/python_intro/blob/master/tutorial-doctest.md

モジュールの補足

- モジュールの利用
  - モジュールのメリット
  - モジュール＝「*.py」
  - importとfromを用いたモジュール読み込み

何のために「モジュール」があるのか？

- 第三者が作成したモジュールを、再利用しやすくするため。
  - 関数だけだと、同一ファイル内でしか再利用できない。
  - モジュールなら、別ファイルでも再利用できる。
- なるべくファイル編集させないようにするため。
  - 人は過ちを侵してしまう。
  - 何気ない編集のつもりが、バグに繋がることも。
  - モジュール（別ファイル）として利用するなら、少なくともファイル編集に伴うバグは発生しない。

モジュールを利用する例

- https://github.com/naltoma/python_demo_module
- ユニットテスト演習で使った例題
  - my_math.py #モジュールとして用意したコード
  - import_case1.py #import例
  - import_case2.py #from+import例

プログラム実行時に自動設定される特殊な変数

- __file__
  - ファイル実行時には「__main__」と設定されるため、if文のTrueブロックが実行される。
    - C言語のmain関数、Javaのmainメソッドに相当。
  - import時には「my_math（モジュール名）」が保存されるため、Trueブロックが実行されていない。

モジュールの動作違い1（main指定なし）

test.py

```
def add(a, b):
    return a + b


print(add(1,2))
```

動作確認

- ファイル実行
  - python test.py #=>3
- import実行
  - python
  - >>> import test
  - >>> 3

モジュールの動作違い2（main指定あり）

test.py

```
def add(a, b):
    return a + b


if __name__ == '__main':
    print(add(1,2))
```

動作確認

- ファイル実行
  - python test.py #=>3
- import実行
  - python
  - >>> import test #=>出力なし
  - >>>

5 Structured types, mutability, and higher-order functions

- 5.1 Tuples
- 5.2 Lists and Mutability
- 5.3 Functions and Objects
- 5.4 Strings, Tuples, and Lists
- 5.5 Dictionaries

5.1 Tuples (タプル型オブジェクト)

- Like strings, **tuples** are ordered sequences of elements. The difference is that the elements of a tuple need not be characters.
- (chap 5.2) Tuples and strings are immutable.
  - 文字列と同様に、タプルは順序のあるシーケンス集合。
  - 文字列と同様に、タプルは変更できないオブジェクト。
  - 文字列と異なり、要素が文字である必要はない。

- コード例

```
>>> t1 = ()
>>> t2 = (1, 'two', 3)
>>> print(t1)
()
>>> print(t2)
(1, 'two', 3)
>>> t2[0]
1
>>> t2[0] = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

タプルの補足1

- 「1個の要素を持つタプル」は特別な書き方が必要。

```
>>> (1)
1
>>> type((1))
<class 'int'>
>>> (1,)
(1,)
>>> type((1,))
<class 'tuple'>
```

- どこで使われる？
  - e.g., 関数の戻り値。
- 他にどういう時に使う？
  - リストよりも高速。
  - 変更を許可したくない場合。
  - 辞書型オブジェクトのキーとして使える。

参考: Dive Into Python 3, 第2章 ネイティブデータ型, http://diveintopython3-ja.rdy.jp/native-datatypes.html

タプルの補足2

- タプル同士の足し算=結合したタプルを生成

```
>>> (1, 2, 3) + (4, 5)
(1, 2, 3, 4, 5)
```

- 複数の変数をまとめて設定

```
>>> x, y, z = (1, 2, 3)
>>> x
1
>>> y
2
>>> z
3
```

- 関数の戻り値

```
def data_analysis(values):
    total = 0
    for value in values:
        total += value
    average = total / len(values)
    return total, average

scores = [50, 70, 90]
total, average = data_analysis(scores)
print(total) # -> 210
print(average) " -> 70.0
```

まとめ

- ユニットテスト
  - 関数が想定通りに動くことを検証しやすくしたり、関数の使い方例示（≒ドキュメントとしての役割）したり、リファクタリングしやすくするために使おう。
- モジュール
  - 必要に応じて__file__を使い、import利用しやすく書こう。
- Tuples
  - 変更できない、順序のあるシーケンス集合。大規模で高速にデータを渡したい場合や、dict型のkeyとして使いたい場合に使おう。