先週の補足2件

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

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

先週の補足2件

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

ユニットテストを用いた開発に慣れよう

from, importによるモジュール読み込みを使えるようになろう。

プログラミング1

(第8回) File I/O, Tuples

1. Chapter 4.6 Files
    1. File I/O (ファイル入出力)
    2. ファイル・ハンドラ
    3. with構文
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
4. 演習
    1. 演習7: doctest, while文
5. 宿題

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2018/prog1/

4 Functions, Scoping, and Abstraction

- 4.1 Functions and Scoping
- 4.2 Specifications
- 4.3 Recursion
- 4.4 Global Variables
- 4.5 Modules
- 4.6 Files

- File Input/Output (File I/O)
  - 「ファイル、アイ、オー」
  - プログラム上でファイルの中身を利用することがFile Input。
  - ファイルへ書き出すことをFile Outputと呼ぶ。

4.6 Files (ファイル)

- ファイルを読み書きするには、
  - Step1:「ファイルハンドラ (file handle)」を準備する。
  - Step2: ハンドラに対して読み書きする。
  - Step3: 読み書きし終えたらハンドラを閉じる。
- コード例

```
name_handle = open('kids', 'w')
for i in range(2):
    name = input('Enter name: ')
    name_handle.write(name + '¥n')
name_handle.close()
```

引数でファイル名とモードを指定。
kidsというファイルに'w'（上書きモード）でアクセスするためのハンドラを用意。

¥=バックスラッシュ。特別な用途を指示。
¥n = 改行
一部環境では「Altキー + ¥」で入力。

「ハンドラ.write()」ハンドラに対して書き込む。
「ハンドラ.close()」ハンドラを閉じる。

代表的なFile I/O操作 (図4.12)

fn = ファイル名

- open()
  - open(fn, 'w'): write(上書きモード)でファイルを用意（あれば中身を消去、なければ新規作成）し、ハンドラを返す。
  - open(fn, 'r'): read(読み込みモード)でファイルを用意し、ハンドラを返す。
  - open(fn, 'a'): append(追記モード)でファイルを用意し、ハンドラを返す。
- ファイルハンドラへの操作
  - fh.read(): サイズ指定がない場合、EOF (End of File) まで読み込み、1つのstr型オブジェクトとして返す。
  - fh.readline(): 改行もしくはEOFまで読み込み、1つのstr型オブジェクトとして返す。
  - fh.readlines(): 行のリストを読み込んで返す。
  - fh.write(s): str型オブジェクトsを書き込む。
  - fh.writelines(S): シーケンス集合Sを書き込む。
  - fh.close(): ハンドラを閉じる。

参考: Python ドキュメント
open(): https://docs.python.org/3/library/functions.html#open
16.2. io — ストリームを扱うコアツール: http://docs.python.jp/3/library/io.html

ファイル操作の推奨方法: with構文

```
#ファイルへ書き込む
with open('spam.txt', 'w') as file:
    file.write('Spam and eggs!¥n')
    file.write('hogege')

#ファイルを読み込む
with open('spam.txt', 'r') as file:
    for raw_line in file:
        line = raw_line.rstrip()
        print(line)
```

- 操作したいファイル名とモードを with 文で指定。
- ブロックの処理が終わると、自動でclose()する。
  ＊途中でエラーになったとしても閉じる。

「str型オブジェクト.rstrip()」
行末の改行文字(¥n)を削除。

参考: Dive Into Python 3, 第11章 ファイル,
http://diveintopython3-ja.rdy.jp/files.html

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。
  - （後で出てくる）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

- File I/O
  - open()でファイルハンドラを用意
  - ハンドラに対して読み書き操作
    - Tips: エディタ（人手）でファイル編集するときと異なり、「ファイルを読んでる最中に書き込む」といった読み書きを同時にすることはできない点に注意。read時にはreadのみ。write時にはwriteのみしよう。read/write混在する場合にはDB利用するほうが楽。
  - 終えたらハンドラを閉じる
    - with構文推奨
- Tuples
  - 変更できない、順序のあるシーケンス集合
  - 高速

演習

演習5: 数当てゲーム1 (大小ヒント付き)を実装してみよう。
演習6: 簡易ガチャ・シミュレータを実装してみよう。

宿題

- 復習: 適宜(これまでの内容)
- 予習: 教科書読み　＊今回は指定なし
  - 余裕がある人
    - 教科書5章の続き
    - バージョン管理: Mercurial, Git
- 復習・予習(オススメ)
  - (paiza) Python入門編1:プログラミングを学ぶ (全9回), https://paiza.jp/works/python3/primer
  - (progate) Python I, II: https://prog-8.com

参考文献

- 教科書: Introduction to Computation and Programming Using Python: With Application to Understanding Data
- Python ドキュメント
  - open(): https://docs.python.org/3/library/functions.html#open
  - 16.2. io — ストリームを扱うコアツール: https://docs.python.org/3/library/io.html
- Dive Into Python
  - 3第11章 ファイル, http://diveintopython3-ja.rdy.jp/files.html
  - 第2章 ネイティブデータ型, http://diveintopython3-ja.rdy.jp/native-datatypes.html