プログラミング1

(第6回) 関数とスコープ、仕様、ユニットテスト、モジュール

1. Chapter 1, 4 Turing Complete
2. Chapter 4.1.2 Keyword Arguments and Default Values
3. Chapter 4.1.3 Scoping
4. Chapter 4.2 Specifications
    1. docstring + doctest (教科書にありません)
5. Chapter 4.5 Modules
6. 演習
    1. 演習1～4: 初めてのレポート
    2. 演習5: if文, 関数の利用
    3. 演習6: while文
7. 宿題

講義ページ: http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/

Chapter 4, 4.1.2, 4.1.3, 4.2, 4.5の補足

4 FUNCTIONS, SCOPING, AND ABSTRACTION
4.1.2 Keyword Arguments and Default Values
4.1.3 Scoping
4.2 Specifications
4.5 Modules

Turing complete (チューリング完全)

＊教科書1章と4章冒頭

- これまでに紹介し終えた機能
  - numbers (数字)
  - assignments (代入)
  - input/output (入出力)
  - comparisons (比較)
  - looping (反復)
- **チューリング完全(=チューリングマシンを再現できる)**
  - プログラミング言語毎の書き易さ・難さはあるが、チューリングマシンを記述可能。

- Church-Turing Thesis (チューリングの提唱)
  - 「もし」ある関数が有限回の操作で計算可能なら、チューリングマシンでプログラム可能であり、それを計算できる。
- Universal Turing Machine (万能チューリングマシン)
  - 0か1を記述できる無限長のテープ（メモリ）があり、テープ上の移動と読み書きする命令を持つ。

コンピュータの原型

参考:
チューリング・マシンとコンピュータ工学:
http://www.slideshare.net/junpeitsuji/ss-57954980

4.1.2 Keywords Arguments and Default Values

```
class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.
```

```
# コード例1
def myrange(start, stop, step=1):
    result = []
    num = start
    while num<stop:
        result.append(num)
        num += step

    return result
```

引数名とデフォルト値の指定

```
# 実行例1
>>> myrange(0, 5)
[0, 1, 2, 3, 4]
>>> myrange(start=0, stop=5)
[0, 1, 2, 3, 4]
>>> myrange(0, 5, step=2)
[0, 2, 4]
```

4.1.3 Scoping (スコープ)

```
# コード例2
>>> def f(x):
...     y = 1
...     x = x + y
...     print('f(x): x = {0}'.format(x))
...     print('f(x): y = {0}'.format(y))
...     return(x)
...
>>> x = 3
>>> y = 2
>>> z = f(x)
f(x): x = 4
f(x): y = 1
>>> print('x = {0}'.format(x))
x = 3
>>> print('y = {0}'.format(y))
y = 2
>>> print('z = {0}'.format(z))
z = 4
```

```
# 左の続き
>>>
>>> def g():
...     print('z = {0}'.format(z))
...
>>> g()
z = 4
```

Stack frame と Name Space

```
def f(x):
    y = 1
    x = x + y
    print('f(x): x = {0}'.format(x))
    print('f(x): y = {0}'.format(y))
    return(x)

x = 3
y = 2
z = f(x)
print('x = {0}'.format(x))
print('y = {0}'.format(y))
print('z = {0}'.format(z))

def g():
    print('z = {0}'.format(z))g()
```

(1) 関数f()の定義

＊コード内部については実行時に作成される。定義時点では関数名のみ。

Stack frame: 1
f()

(2) コードの実行

Stack frame: 1
f(), x=3, y=2

(3) 関数実行(関数呼び出し)

(3-1) 関数呼び出し直前

Stack frame: 1
f(), x=3, y=2, z

(3-2) 関数実行,return直前

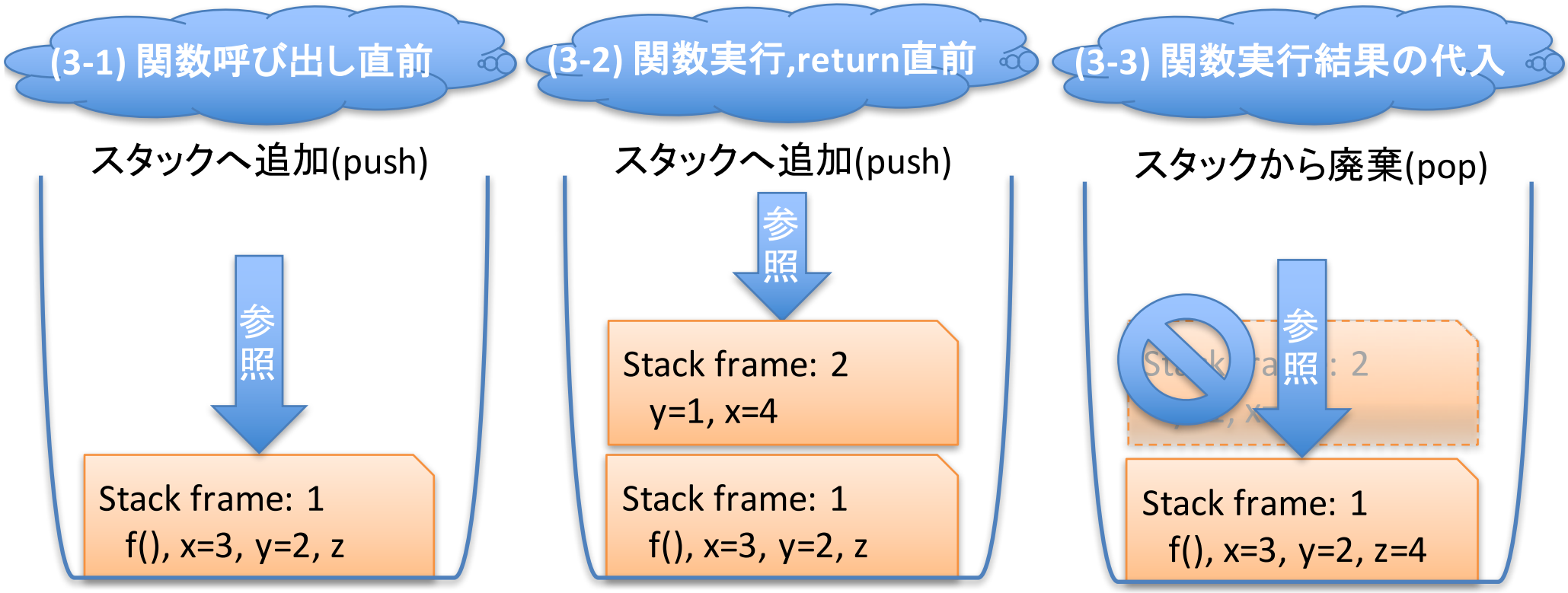
Stack frame: 2
y=1, x=4

(3-3) 関数実行結果の代入

Stack frame: 1
f(), x=3, y=2, z=4

参考: 基本的なデータ構造,
https://ja.wikipedia.org/wiki/データ構造

「スタック」構造=Last-in, First-out (LIFO)



Stack Frame のポイント

- トップレベル（インタプリタ起動時、もしくはスクリプトファイル実行時の最初のブロック）では、全ての関数名・変数名をトレースし、最初のスタックフレームで紐付けされる。
- 関数が呼び出されると、新しいスタックフレームが作られる。
  - ここでの処理は、スコープ外にあるスタックフレームには影響を及ぼさない。（例外あり）
  - 関数が終了すると、スタックフレームは廃棄(pop)される。
  - 現スタックフレームに記録されていない名前が参照されると、「呼び出し元のスタックフレーム」を検索する。この遡り検索をトップレベルまで繰り返しても見つからない場合、NameErrorとなる。

ループ処理の例 (2.4節の改良版)

http://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/loop.py

```
# スライムのHPが0より大きい間タコ殴りにするゲーム

import random

def encount_enemy():
    hitpoint = random.randint(3,7)
    return hitpoint

enemy_hitpoint = encou                my()
print('スライムに遭遇した                enemy_hitpoint,'')

while (enemy_hitp
    attack = random
    enemy_hitpoint
    print('あなたの攻
HP=', enemy_hitpo

print('スライムを倒
```

4週目のwhile文コード例

(1) 関数encount_enemy()で参照してる「random」は、この関数内では定義されていない。

(2) 関数実行中のスタックフレームに存在しないため、トップレベルのスタックフレームを参照する。

(3) トップレベルのスタックフレームには、「import random」で読み込んだrandomモジュールが登録されており、これを関数encount_enemy()でも利用する。

4.2 Specifications (仕様, ドキュメント)

- コメント文
  - 「#」から後の文
- docstring形式によるコメント
  - 「"""～"""」で囲った、複数行に渡るコメント。
  - インデントでブロックを揃えること。
  - 簡易ドキュメントとしての側面

```
% curl https://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/week6_doctest.py -o week6_doctest.py
% pydoc3 -w week6_doctest
% open week6_doctest.html
```

pydoc3:
ドキュメンテーション・ツール
＊引数はモジュール名
（ファイル名ではない）

参考: Example Google Style Python Docstrings
http://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html

4.5 Modules (モジュール, 部品)

- コードを保存したファイル＝モジュール
- 使用例1

```
% python3
>>> import week6_doctest
>>> week6_doctest.add(1, 2)
3
```

自作モジュールをimportで読み込む

「モジュール名.**」でモジュール内の部品を利用。

- 使用例2

```
% python3
>>> import week6_doctest as wd
>>> wd.add(1, 2)
3
```

モジュール名に別称を付ける

- 使用例3

```
% python3
>>> from week6_doctest import *
>>> add(1, 2)
3
```

モジュール内の部品を指定して読み込む。「*」は全ての部品。

- 読み込んだモジュールはhelp()で簡易ドキュメントを参照できる。

```
>>> import week6_doctest as wd
>>> help(wd)
```

pydoc3で読めるドキュメントと、内容は同じ。

doctestによるユニットテスト (教科書にありません)

- ユニットテスト (Unit Test, 単体テスト)
  - 関数が想定通りに機能するかを検証。
  - print出力して目視確認ではなく、「想定した入力を与えたら、想定した結果が得られること（成功or失敗）」を確認。
- doctestによるユニットテスト
  - 実行できるドキュメント（docstring）
  - 例
    - 通常のスクリプト処理

      % python3 week_doctest.py
    - ユニットテスト処理

      % python3 week_doctest.py -v

通常処理ではユニットテストは実行されない。（開発中に確認するもの）

doctest実行例

```
oct:tnal% python3 week6_doctest.py  -v
Trying:
    add(1, 2)
Expecting:
    3
ok
Trying:
    add(-1, 3)
Expecting:
    2
ok
Trying:
    add(0, 0.5)
Expecting:
    0.5
ok
(右に続く)
```

テスト1

テスト2

テスト3

2つのアイテムにある全てのテストをパスした（想定通りだった）
・1つのテストが__main__にあった。
・2つのテストが__main__.addにあった。

```
(左の続き)
2 items passed all tests:
   1 tests in __main__
   2 tests in __main__.add
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
```

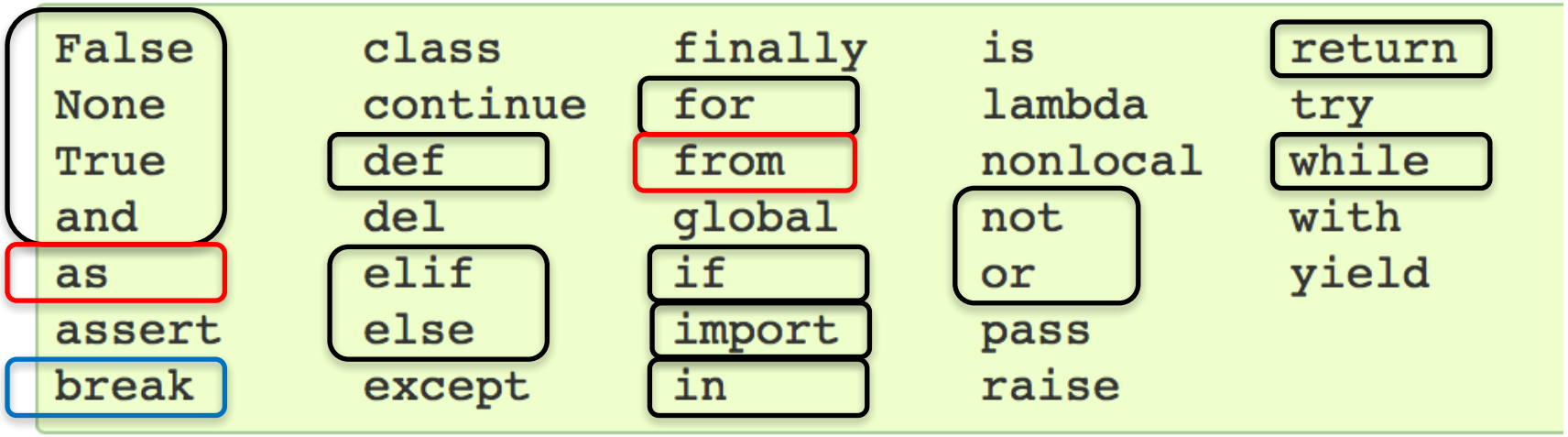
doctestの注意点

- 「インタプリタでの出力と完全一致」じゃないと「想定通り」と見做してくれない。
  - 「1」と「1 」は違う
  - [1, 2]と[1,2]は違う
    - doctest的にはスペースの有無も厳密にチェック。
    - str型オブジェクトとしての判定。

Reserved words, 予約語

https://goo.gl/rEzdAN

- 一覧（赤丸は今回出てきた予約語）



まとめ

- スタックフレームとスコープに基づく名前空間の参照
  - 今書いてるスコープで参照されるスタックフレームはどれか？
  - 大原則
    - 関数を呼び出すと新しくスタックフレームが追加される。
    - 関数の処理を終えると、スタックフレームは破棄される。
    - 名前空間は現スタックフレームから遡って検索する。
- from, import によるモジュール読み込み
- docstringによるドキュメンテーション
  - 自由に記述できるが、できるだけガイドに従おう。
- doctestによるユニットテスト
  - 実行できるドキュメント

演習6まで終わったペアは、進捗報告してから他ペアの様子を眺めてみよう。（第2のobserverしよう）

演習

演習1～4: 初めてのレポート

演習5: if文, 関数の利用

演習6: while文の利用

補足1

- ペアプログラミングを始める前に
  - 記入漏れ
    - 「実施日」と「報告者」
  - 前回の復習確認
    - 「何をやったっけ？」
    - 「これはこうやれば良いんだっけ？」

補足2

- ペアプログラミングのやり方
  - 7ステップ
    - 作業を決める
    - 最初の目標を決める
    - パートナーを頼りにし、支えてやる
      - driver: 仕事を終わらせることに専念
      - observer: 横から観察し、疑問・改善・簡潔化など大局的な問題について考える
    - 喋る
      - 「一人で悩む」のは十秒程度に留める
      - 一緒に相談しながら考える練習
    - お互い何をやっているか把握する
      - 頻繁に同期をとる
    - 喜ぶ
    - 交代する

分業ではない（observer=観察しながら気づいたことをコメント）

二人で2,3分考えても分からない場合には、手を上げて質問しよう

演習

- 演習1～演習4: 初めてのレポート
  - https://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/week2-ex.html
  - レポートの作成手順は授業ページを参照
- 演習5: if文と関数
  - https://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/week3-ex.html
- 演習6: while文
  - https://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/week4-ex.html
- ペア・プログラミング
  - https://ie.u-ryukyu.ac.jp/~tnal/2016/prog1/week2-pair-programming.html
  - driver, observer (navigator)

宿題

- 復習: 適宜(これまでの内容)
  - レポート課題2「コード読解」　＊講義ページ参照。
    - レポート課題1の良レポートの「良さ」を真似よう。
- 予習: 教科書読み
  - 4章
    - 4.3 Recursion
    - 4.4 Global Variables
- 復習・予習(オススメ): paiza
  - プログラミングスキルチェック　＊レベル設定のある課題集
    - https://paiza.jp/challenges/info

参考文献

- 教科書: Introduction to Computation and Programming Using Python, Revised And Expanded Edition
- Python 3.5.1 documentation, https://docs.python.org/3.5/index.html
- チューリング・マシンとコンピュータ工学, http://www.slideshare.net/junpeitsuji/ss-57954980
- doctest — Test interactive Python examples, https://docs.python.org/3/library/doctest.html
- Example Google Style Python Docstrings, http://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html