

プログラミング1

(第5回) ループ処理(for文)、range()関数とリストによるシーケンス 集合表現

1. 演習・課題への取り組み方、レポートの書き方のデモ
2. Chapter 3.2 For Loops
 1. もう一つのループ処理
 2. シーケンス集合とコード例
3. Chapter 3.4 A Few Words About Using Floats
 1. 浮動小数点数の取り扱い
4. 演習
 1. 演習1~4: 初めてのペア・プログラミング
 2. 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
5. 宿題

KISS原則を踏まえて小さく作ろう。分割の仕方を考えよう。

シーケンス集合に対する反復処理を書けるようになろう。

小数を使う際には丸め誤差と桁あふれに注意。

プログラミング1

(第6回) デバッグ実行、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 (恐らく時間なし)
7. 宿題

講義ページ: <http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/>

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>

プログラミング1

(第6回) デバッグ実行、関数とスコープ、仕様、ユニットテスト、モジュール

コンピュータの原型である**万能チューリングマシン**について調べてみよう。

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 (恐らく時間なし)
7. 宿題

講義ページ: <http://ie.u-ryukyu.ac.jp/~tnal/2017/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

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]
```

プログラミング1

(第6回) デバッグ実行、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 (恐らく時間なし)
7. 宿題

関数の引数には
デフォルト値を設定できる！

講義ページ: <http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/>

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

Stack frame: 1
f()

* 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。

(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

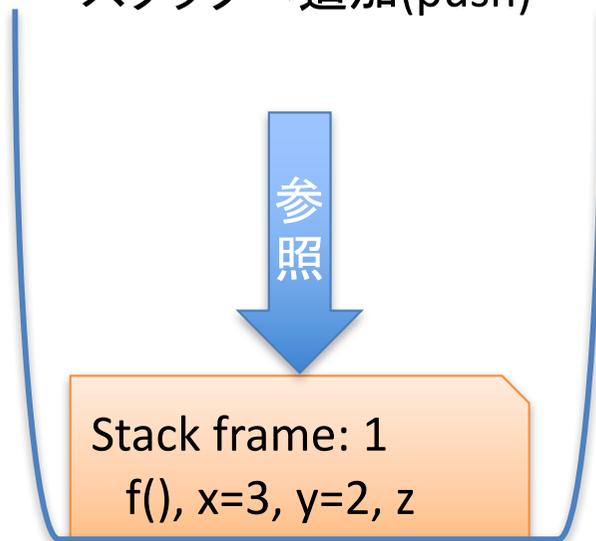
(3-3) 関数実行結果の代入

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

「スタック」構造=Last-in, First-out (LIFO)

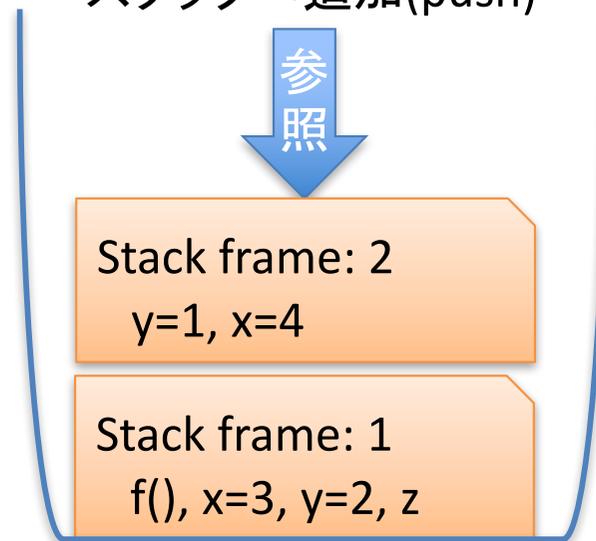
(3-1) 関数呼び出し直前

スタックへ追加(push)



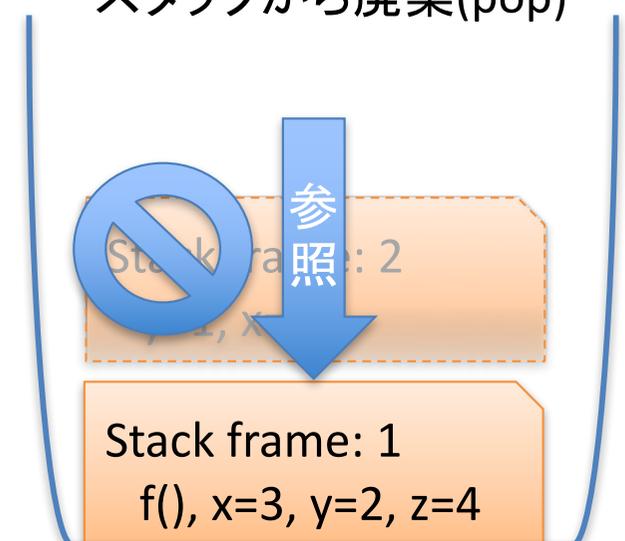
(3-2) 関数実行,return直前

スタックへ追加(push)



(3-3) 関数実行結果の代入

スタックから廃棄(pop)



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。この逆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/2017/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 = encount_enemy()  
print('スライムに遭遇した。敵のHPは', enemy_hitpoint, '！')
```

```
while (enemy_hitpoint > 0):  
    attack = random.randint(1, 3)  
    enemy_hitpoint -= attack  
    print('あなたの攻撃で、敵のHPは', enemy_hitpoint, 'になりました。')  
print('スライムを倒した。')
```

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。

プログラミング1

(第6回) デバッグ実行、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 (恐らく時間なし)
7. 宿題

関数を呼び出しとスタックフレームの追加、名前空間の遡り参照を理解しよう。

講義ページ: <http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/>

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/2017/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

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 week6_doctest.py`
 - ユニットテスト処理
`% python3 week6_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
```

テスト1

```
Trying:
```

```
    add(-1, 3)
```

```
Expecting:
```

```
    2
```

```
ok
```

テスト2

```
Trying:
```

```
    add(0, 0.5)
```

```
Expecting:
```

```
    0.5
```

```
ok
```

テスト3

(右に続く)

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。

プログラミング1

(第6回) デバッグ実行、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 (恐らく時間なし)
7. 宿題

ドキュメンテーション
とユニットテストを用
いた開発に慣れよう。

講義ページ: <http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/>

4.5 Modules (モジュール, 部品)

モジュール内の部品を指定して読み込む。「*」は全ての部品。

- コードを保存したファイル = モジュール

- **使用例1**

自作モジュールをimportで読み込む

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

「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
```

- 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。

プログラミング1

(第6回) デバッグ実行、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 (恐らく時間なし)
7. 宿題

from, importによるモジュール読み込みを
使えるようになる。

講義ページ: <http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/>

Reserved words, 予約語

<https://goo.gl/rEzdAN>

- 一覧(赤丸は今回出てきた予約語)

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

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
 - 今書いてい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

プログラミング1

(第6回) デバッグ実行、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

コンピュータの原型である**万能チューリングマシン**について調べてみよう。

2. Chapter 4.1.2 Keyword Arguments and Default Values

関数の引数には**デフォルト値**を設定できる！

3. Chapter 4.1.3 Scoping

関数を呼び出しと**スタックフレーム**の追加、**名前空間**の遡り参照を理解しよう。

4. Chapter 4.2 Specifications

1. docstring + **doctest** (教科書にありません)

ドキュメンテーションと**ユニットテスト**を用いた開発に慣れよう。

5. Chapter 4.5 Modules

from, importによるモジュール読み込みを**使える**ようになろう。

6. 演習 (恐らく時間なし)

7. 宿題

講義ページ: <http://ie.u-ryukyu.ac.jp/~tnal/2017/prog1/>

演習5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: 初めてのペア・プログラミング

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

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):
 - 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