Python勉強会@HACHINONE

第5章

構造型、可変性と高階関数

お知らせ

Python勉強会@HACHINOHEでは、ジョン・V・グッターグ『Python言語による
プログラミングイントロダクション』近代科学社、2014年をみんなで勉強しています。

この本は自分で読んで考えて調べると力が付くように書かれています。

自分で読んで考えて調べる前に、このスライドを見るのは、いわばネタバレを
聞かされるようなものでもったいないです。

是非、本を読んでからご覧ください。

オブジェクトの内部構造

Python勉強会@HACHINOHE

- スカラ型
  - 内部構造がない: 1つしかデータを持たない
  - intやfloat
- 構造型、非スカラ型
  - 内部構造を持つ: 複数のデータを持つ
  - strなど: 'Hello, world!'[1] => 'e'

コレクションのための3つの構造型

Python勉強会@HACHINOHE

	可変性	例
タプル tuple	不変 immutable	`(1, 'two', 3)`
リスト list	可変 mutable	`['I did it all', 4, 'love']`
辞書 dict		`{ 'Jan':1, 'Feb':2, 'Mar':3, ... }`

タプル

Python勉強会@HACHINOHE

- いろいろな型を入れられるコレクション
  - 1つの要素の場合は(1,)のように書く

```
t1 = (1, 'two', 3)
t2 = (t1, 3.25)
print t2              # ((1, 'two', 3), 3.25)
print (t1 + t2)       # (1, 'two', 3, (1, 'two', 3), 3.25)
                      # 「+」はリストを結合し、
                      # 新たなリストを生成する。
                      # つまり、t1とt2は変化を受けない
print (t1 + t2)[3]    # (1, 'two', 3)
print (t1 + t2)[2:5] # (3, (1, 'two', 3), 3.25)
```

- for文で要素を1つずつ取り出して扱える

```
for i in t1
    print i
```

```
1
two
3
```

20と100の公約数とその総和

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
def findDivisors (n1, n2):
    """n1とn2を正のintとする。
       n1とn2のすべての公約数からなるタプルを返す"""
    divisors = () # 公約数を入れるタプルを空で初期化
    for i in range(1, min (n1, n2) + 1):
        if n1 % i == 0 and n2 % i == 0: # 公約数(n1もn2も割り切れる数)なら
            divisors = divisors + (i,) # タプルdivisorsに追加
                                       # 実際には新たにタプルが生成される
    return divisors

divisors = findDivisors(20, 100)
print divisors # (1, 2, 4, 5, 10, 20)
total = 0
for d in divisors:
    total += d
print total # 42
```

新たなタプルの生成

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
def findDivisors (n1, n2):
    """n1とn2を正のintとする。
       n1とn2のすべての公約数からなるタプルを返す"""
    divisors = () # 公約数を入れるタプルを空で初期化
    for i in range(1, min (n1, n2) + 1):
        if n1 % i == 0 and n2 % i == 0: # 公約数(n1もn2も割り切れる数)なら
            print 'id=', id(divisors)
            divisors = divisors + (i,) # タプルdivisorsに追加
                                       # 実際には新たにタプルが生成される
    return divisors

divisors = findDivisors(20, 100)
print divisors # (1, 2, 4, 5, 10, 20)
total = 0
for d in divisors:
    total += d
print total # 42
```

```
id= 4536954960
id= 4538340048
id= 4538303912
id= 4538362288
id= 4538382480
id= 4538158064
```

最大最小公約数

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
def findExtremeDivisors (n1, n2):
    """n1とn2を正のintとする。
       n1とn2の1より大きい最小の公約数と最大公約数からなるタプルを返す。
      公約数がない場合は(None, None)を返す。"""
    divisors = ()
    minVal, maxVal = None, None
    for i in range(2, min(n1, n2) + 1):
        if n1 % i == 0 and n2 % i == 0:
            if minVal == None or i < minVal:
                minVal = i
            if maxVal == None or i > maxVal:
                maxVal = i
    return (minVal, maxVal)

minDivisor, maxDivisor = findExtremeDivisors(100, 200) # 多重代入
print minDivisor, maxDivisor
```

最大最小公約数

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
def findExtremeDivisors (n1, n2):
    """n1とn2を正のintとする。
       n1とn2の1より大きい最小の公約数と最大公約数からなるタプルを返す。
      公約数がない場合は(None, None)を返す。"""
    minVal, maxVal = None, None
    for i in range(2, min(n1, n2) + 1):
        if n1 % i == 0 and n2 % i == 0:
            if minVal == None: # 最初に成り立ったときが最小公約数
                minVal = i
            maxVal = i # 最後に成り立ったときが最大公約数
    return (minVal, maxVal)

minDivisor, maxDivisor = findExtremeDivisors(100, 200)
print minDivisor, maxDivisor
```

リスト

Python勉強会@HACHINOHE

- リストのリテラル `L = ['I did it all', 4, 'love']`
- リストの要素へのアクセス `リスト[添字]`
- 例

```
L = ['I did it all', 4, 'love']
for i in range(len(L)):
    print L[i]
```

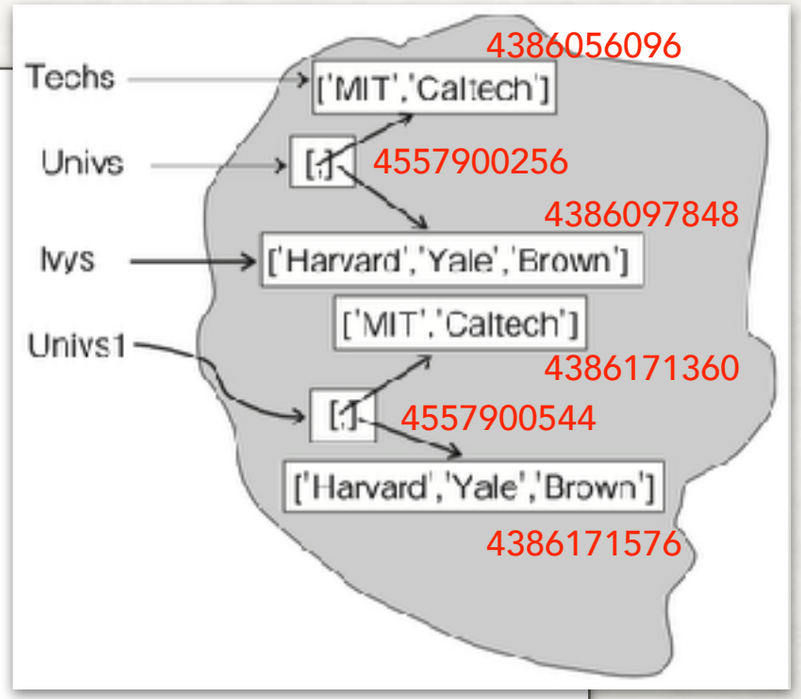
```
L = ['I did it all', 4, 'love']
for l in L:
    print l
```

- メソッド
  - リストを改変する: append、insert、extend、remove、pop、sort、reverse
  - リストを改変しない: count、index、+

オブジェクト同一性

Python勉強会@HACHINOHE

- 同じオブジェクトなのか、たまたま2つのオブジェクトの値が一致しているのか。id(オブジェクト)で確認できる



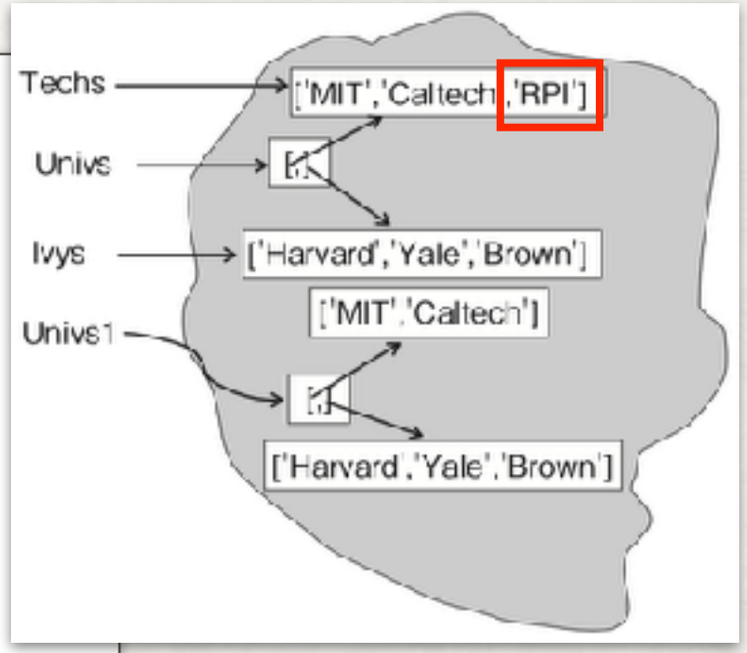

```
# -*- coding: utf-8 -*-
Techs = ['MIT', 'Caltech']
Ivys = ['Harvard', 'Yale', 'Brown'] # アイビー・リーグ

Univs = [Techs, Ivys]
Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]
print 'Univs =', Univs
print 'Univs1 =', Univs1

print Univs == Univs1                    # True。値は一致
print id(Univs) == id(Univs1)            # False。別オブジェクト
print 'UnivsのIdは' + str(id(Univs))   # UnivsのIdは4557900256
print 'Univs1のIdは' + str(id(Univs1)) # Univs1のIdは4557900544
print "Univs[0]とUnivs[1]のIdは" + str(id(Univs[0])) + "と" + str(id(Univs[1]))
# Univs[0]とUnivs[1]のIdは4386056096と4386097848
print "Univs1[0]とUnivs1[1]のIdは" + str(id(Univs1[0])) + "と" + str(id(Univs1[1]))
# Univs1[0]とUnivs1[1]のIdは4386171360と4386171576
```

参照

Python勉強会@HACHINOHE

- TechsとUnivsは同じリストを参照(エイリアシング)。Techsを操作するとUnivsも変化を受ける

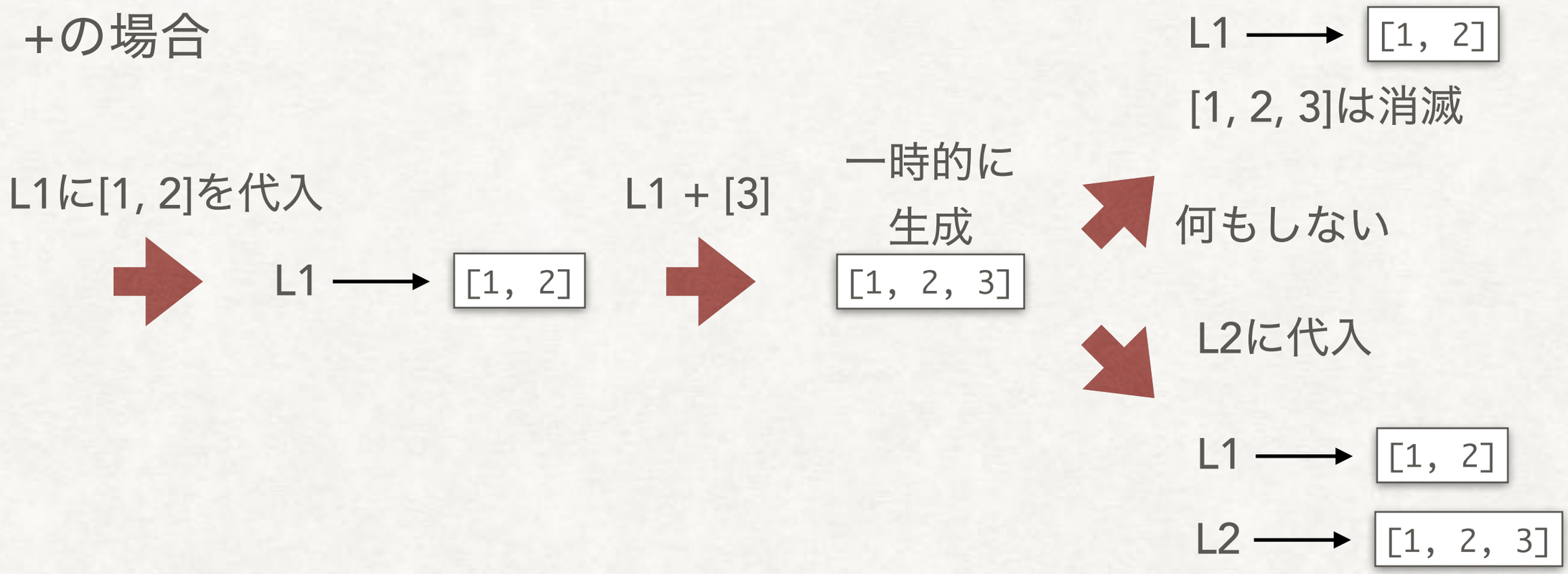
```
# -*- coding: utf-8 -*-
Techs = ['MIT', 'Caltech']
Ivys = ['Harvard', 'Yale', 'Brown']

Univs = [Techs, Ivys]
Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]
print 'Univs =', Univs
# Univs = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]
print 'Univs1 =', Univs1
# Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]

Techs.append('RPI') # レンセラー工科大学
print 'Univs =', Univs
# Univs = [['MIT', 'Caltech', 'RPI'], ['Harvard', 'Yale', 'Brown']]
print 'Univs1 =', Univs1
# Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]
```



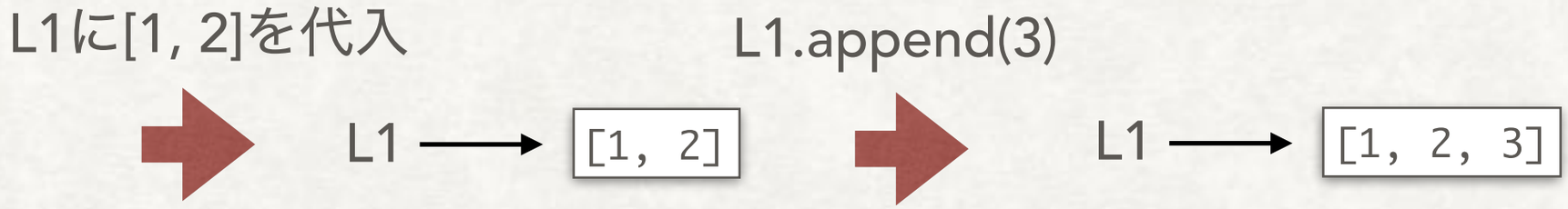
可変性

Python勉強会@HACHINOHE

- +の場合



- appendの場合



クローン

Python勉強会@HACHINOHE

- 反復中にリストを改変すると...
- クローンを作るにはスライシングかlist()

```
# -*- coding: utf-8 -*-
def removeDups(L1, L2):
    """L1とL2はリスト。
       L1からL2の要素を取り除く"""
    newL1 = L1[:] # リストのクローンを作る。L1[:]はlist(L1)でもいい
    for e1 in newL1:
        if e1 in L2:
            L1.remove(e1) # 要素を取り除くとL1がズレる

L1 = [1, 2, 3, 4]
L2 = [1, 2, 5, 6]
removeDups(L1, L2)
print 'L1 =', L1 # 青字の部分がないと[2, 3, 4]
```

ディープ・コピー

Python勉強会@HACHINOHE

- 参照先まで複製(ディープ・コピー)するには、copyモジュールのdeepcopy

```
# -*- coding: utf-8 -*-
import copy
Techs = ['MIT', 'Caltech']
Ivys = ['Harvard', 'Yale', 'Brown']

Univs = [Techs, Ivys] 参照を含んだリスト
Univs1 = Univs # Univsと同じ実体を参照
Univs2 = list(Univs) # Univsをクローン
Univs3 = copy.deepcopy(Univs) # Univsをディープ・コピー

Univs.append(['Tokyo', 'Kyoto'])
Techs.append('RPI')

print Univs
# Univs = [['MIT', 'Caltech', 'RPI'], ['Harvard', 'Yale', 'Brown'], ['Tokyo', 'Kyoto']]
print Univs1
# Univs1 = [['MIT', 'Caltech', 'RPI'], ['Harvard', 'Yale', 'Brown'], ['Tokyo', 'Kyoto']]
print Univs2
# Univs2 = [['MIT', 'Caltech', 'RPI'], ['Harvard', 'Yale', 'Brown']]
print Univs3
# Univs3 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]
```

リスト内包

Python勉強会@HACHINOHE

- 順序型の各要素に対して演算を行なったリストを生成

```
# -*- coding: utf-8 -*-
# [1, 2, ..., 6]を2乗したリスト
L = [x**2 for x in range(1, 7)]
print L
# [1, 4, 9, 16, 25, 36]

mixed = [1, 2, 'a', 3, 4.0]
# mixedの整数だけを2乗したリストを表示
print [x**2 for x in mixed if type(x) == int]
# [1, 4, 9]
```

関数オブジェクト

Python勉強会@HACHINOHE

- Pythonの関数は第1級オブジェクト
  - 関数を変数に代入、関数を引数に指定したりできる

```
# -*- coding: utf-8 -*-
# 高階(関数を引数にとる)関数applyToEach
def applyToEach(L, f):
    """Lはリスト、fは関数。
       Lのそれぞれの要素を、fを適用したものに書き換える"""
    for i in range(len(L)):
        L[i] = f(L[i])

L = [1, -2, 3.33]
print 'L =', L                  # L = [1, -2, 3.33]
print 'Lの各要素にabsを適用'
applyToEach(L, abs)
print 'L =', L                  # L = [1, 2, 3.33]
print 'Lの各要素にintを適用'
applyToEach(L, int)
print 'L =', L                  # L = [1, 2, 3]
```

map関数

Python勉強会@HACHINOHE

- Pythonには、高階関数map(写像)が用意されている

```
map(関数，リスト1，リスト2，...)
```

```
# -*- coding: utf-8 -*-
print map(abs, [1, -2, 3.33]) # [1, 2, 3.33]

L1 = [1, 28, 36]
L2 = [2, 57, 9]
print map(min, L1, L2) # [1, 28, 9]
```

順序型に共通する演算

Python勉強会@HACHINOHE

- seq[i]: i番目の要素
- len(seq): 要素の個数
- seq1 + seq2: seq1とseq2を連結
- n * seq: seqをn個連結
- seq[start:end]: スライス
- e in seq: eがseqに含まれるか
- e not in seq: eがseqに含まれないか
- for e in seq: seqの要素を反復してeとして取り出す
- 更に文字列にはcount、find、...がある(図5.6)

辞書

Python勉強会@HACHINOHE

- 辞書は、キーと値のコレクション
  - 内部ではハッシュを使用

```
辞書 = { キー:値，キー:値，...}
```

- キーで値を参照できる

```
辞書[キー]
```

```
# -*- coding: utf-8 -*-
# 数字をキーにした名前と、名前をキーにした数字の辞書
monthNumbers = {
    'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May':5,
    1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May'
}

print '3番目の月は' + monthNumbers[3]
dist = monthNumbers['Apr'] - monthNumbers['Jan']
print 'AprとJanは' + str(dist) + 'つ離れている'
```

- 新しい要素を追加するには

```
辞書[新しいキー] = 値
```

辞書の要素の反復

Python勉強会@HACHINOHE

- 辞書にfor文を使うと、キーが取り出される

```
# -*- coding: utf-8 -*-
monthNumbers = {
    'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May':5
}
for e in monthNumbers:
    print e

monthNumbers = {
    1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May'
}
for e in monthNumbers:
    print e
```

```
Jan
Apr
Mar
Feb
May
```

```
1
2
3
4
5
```

辞書に関する命令

Python勉強会@HACHINOHE

- len(d): 要素の個数
- d.keys(): キーのリスト。Python 3ではdict_keys型
- d.values(): 値のリスト。Python 3ではdict_values型
- k in d: キーkがdに含まれるか
- d[k]: キーがkの要素の値
- d.get(k, v): キーkがdに含まれればその値、なければv
- d[k] = 値: キーkの値の書き換え、あるいは新規登録
- del d[k]: キーkの要素を削除
- for k in d: dのキーをひとつずつkに取り出して反復

まとめ

Python勉強会@HACHINOHE

- コレクションの使い方、メソッドなど
  - タプル
  - リスト
  - 辞書
- 可変性
- 高階関数を利用し、コレクションに対して任意の関数を演算する