2018年航空宇宙情報システム学第2
第2部「プログラミングと数値計算」

第5回 Python入門 4
～辞書・タプル～

2018年6月5日

次週休講のお知らせ

- 都合により、6月12日は休講になります。

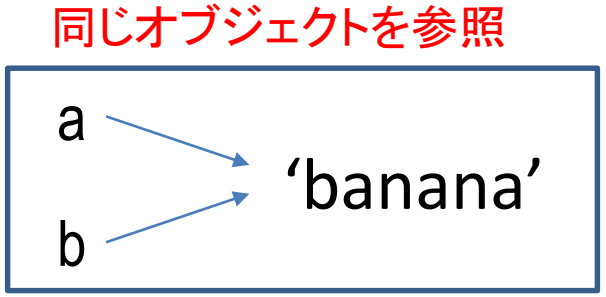
スケジュールは以下のように変更します。

- 6/12 : 休講
- 6/19 : Python 入門５　（ファイル入出力）
- 6/26 : 数値計算入門１（行列・ベクトル演算）
- 7/3 :数値計算入門２（常微分方程式）
- 7/10 :数値計算入門３（最小二乗法、最適化）
- 7/17 :数値計算入門4（考え中）

前回の補足と残り (10章リスト)

オブジェクトと値

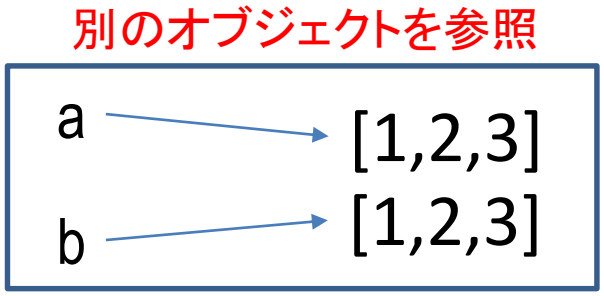
- 同じ文字列を2つの変数に代入




```
>>> a = ‘banana’
>>> b = ‘banana’
```

この場合は、同じオブジェクトを指している

- リストの場合はどうか？



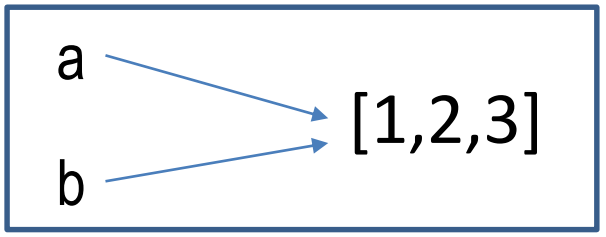

```
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> a is b
False          # オブジェクトとしては異なる
>>> a == b
True           # 値としては等しい
```

オブジェクトの別名 (Aliasing)

- リストを他の変数に代入すると..

```
>>> a = [1,2,3]
>>> b = a
>>> b is a
True
```

  – この場合、a と b は同じリストオブジェクトへの参照、あるいは別名となっている



- 同じオブジェクトへの別名なので、一方の要素を変更するともう一方も影響を受ける

```
>>> a[0] = -1  # 先頭を変更
>>> b
[-1, 2, 3]
```

この後、
>>> a = [1,2,3]
としたら、b はどうなるか？

（補足）リストのコピー

- リストのコピー（クローン）を作るには？

  >>> a = [1,2,3]

- 方法1：スライスを使う

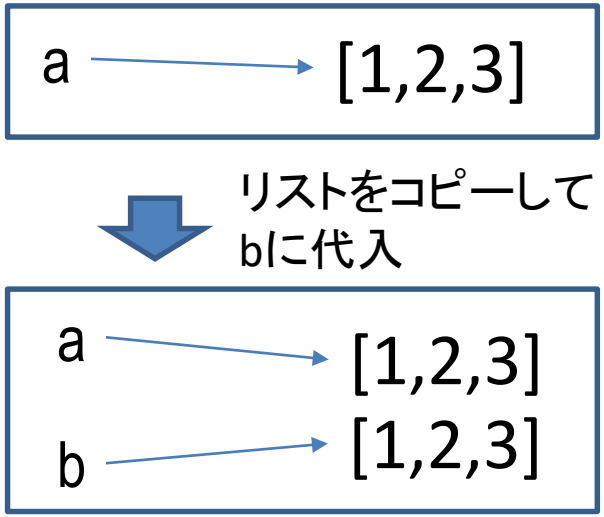
  >>> b = a[:]

- 方法2：list 関数を使う

  >>> b = list(a)

- 方法3：copyメソッドを使う

  >>> b = a.copy()



- ただし、リストの要素がリスト（リストが入れ子）の場合は注意が必要
  - 「浅い」コピー v.s.「深い」コピー

関数の引数にリストを渡す

- 関数の引数にリストを渡すと、コピーではなくて参照が渡される

```
>>> def square(lst):
...     for i in range(len(lst)):
...         lst[i] = lst[i]*lst[i]   # 値を2乗して変更
...
>>> l = [1.0, 2.0, 3.0]
>>> square(l)
>>> l
[1.0, 4.0, 9.0] # リスト l の中身が変更されている
```

11章 辞書

辞書(dictionary)とは

- リストを一般化したもの
  - リストのインデックス : 整数でなければならない
  - 辞書のインデックス(キー) :どんな型でも良い
- “辞書”はキー(key)から値(value)へのマップである
  - キーと値の組み合わせをアイテムと呼ぶ
- 例 :英西単語辞書

```
>>> eng2sp = dict()    # 空の辞書が作成される
>>> eng2sp[ ‘one’ ] = ‘uno ‘ #アイテムの追加
#別の作り方(キーと値の組をコロン区切りで指定)
>>> eng2sp = {’one’: ’uno’, ’two’: ’dos’, ’three’: ’tres’}
>>> eng2sp[ ‘two’ ]
 ‘dos’
```

辞書の基本

- 存在しないキーを指定するとエラーが出る

```
>>> eng2sp['four']
KeyError: 'four'
```

- len関数は、アイテム(キー-値の組)の数を返す

```
>>> len(eng2sp)
3
```

- in 演算子は、キーに含まれるか否かを判定

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
>>> 'uno' in eng2sp.values() # valuesメソッドを使う
True
```

使用例:各文字の出現回数調査

- 文字列の中の各文字の出現回数を求める関数

```
def histogram(s):
    d = dict()    # 空の辞書（カウンタ） d を作成
    for c in s:    # 文字列sから1文字ずつ取り出す
        if c not in d:    # 文字cが初めて出現した場合
            d[c] = 1    # 初期化
        else:                # 2回目以降
            d[c] += 1  # カウンタをインクリメント
    return d    # 集計結果を返す
```

- 使い方

```
>>> h = histogram('brontosaurus')
>>> print h
```

getメソッドの利用

- 上のhistgram関数の例は、辞書型のgetメソッドを使うと簡潔に書くことができる

```
def histogram(s):
    d = dict()
    for c in s:
        d[c] = d.get(c,0) + 1
    return d
```

  - 辞書型オブジェクト.get(key,default) は、辞書オブジェクトにkey がキーとして存在していればその値を、存在していなければ、代わりにdefaultを返す

繰り返し処理と辞書型

- 辞書 h の各アイテムのキーと値を表示する関数

```
def print_hist(h):
    for c in h:     # 辞書hに含まれるキーを1つずつ取得
        print(c, h[c])
```

- 実行例:

```
>>> h = histogram('parrot')
>>> print_hist(h)
a 1
p 1
r 2
:   :
```

逆引き辞書の作成

- 出現回数から文字を調べる関数 invert_dict

```
def invert_dict(d):
    inverse = dict()
    for key in d:
        val = d[key]
        if val not in inverse:
            inverse[val] = [key]
        else:
            inverse[val].append(key)
    return inverse
```

- 赤字の部分を、前述のgetメソッドを使って1行で書き換えることも可能

（応用）辞書型をメモに使う Fibonacci関数の場合

$$\text{fibonacci}(n) = \begin{cases} 0 & (\text{if } n = 0) \\ 1 & (\text{if } n = 1) \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & (\text{otherwise}) \end{cases}$$

- 再帰呼び出しを使って素朴に実装すると..

```
def fibonacci (n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

問題点　：　例えば、fibonacci(99) の後に fibonacci(100)を呼び出すと同じ計算を繰り返して非効率

（応用）辞書型をメモに使う Fibonacci関数の場合（続き）

- 解決策：辞書を使って、過去の計算結果を保存（キャッシュ）しておけば良い

```
known = {0:0, 1:1}  # fibonacci(n)の計算結果の記録
def fibonacci(n):
    if n in known:  # 辞書の中にあればそれを返す
        return known[n]
    res = fibonacci(n-1) + fibonacci(n-2)
    known[n] = res  # 計算結果を辞書に登録
    return res
```

fibonacci(99) を計算した後に fibonacci(100)を呼び出すと一瞬で答えが出る

グローバル変数

- 前の例の 辞書 known はグローバル変数
  - プログラムのどこからでもアクセス可能
  - c.f. ローカル変数はその関数の中でのみ有効
- グローバル変数の典型的利用:「フラグ変数」
  - デバッグのときに便利

```
verbose = True  #フラグ変数（グローバル変数）

def example1():
    if verbose: # フラグ変数を参照
            print('Running example1 ')
```

グローバル変数(続き)

- グローバル変数への再代入は注意が必要

```
been_called = False    # グローバル変数
def example2():
    global been_called  # 代入にはこの宣言文が必要 !
    been_called = True # グローバル変数への再代入

count = 0
def example3():
    global count  # グローバル変数として宣言
    count += 1
```

- ただし、変更可能なリストや辞書オブジェクトでは、global宣言不要
  - 非常にややこしい。試行錯誤で覚えよう。

12章 タプル

タプル（Tuples）

- リストと似ているが、変更不可
- コンマ区切りで値を列挙して生成

```
>>> t = 'a', 'b', 'c', 'd', 'e'
>>> t = (‘a’, ‘b’, ‘c’, ‘d’, ‘e‘) # カッコで囲むこともある
>>> t1 = ‘a’, # 1要素だけのタプル（カンマが重要）
>>> t = tuple() # 空のタプル
>>> t = () # これも空のタプル
```

- リストや文字列からタプルを生成

```
>>> tuple('lupins')
>>> tuple(range(1,5))
```

タプルの操作

- リストに使える操作の多くはタプルにも有効

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print t[0]    #インデックスを使ってアクセス
>>> print t[1:3] # スライスも使える
```

- しかし、要素の変更は不可

```
>>> t[0] = 'A'    # エラーが出る
```

- 新しいタプルを作って代入することは可能

```
>>> t = ('A',) + t[1:]
```

タプルの代入

- タプルを使えば複数の変数にまとめて代入できる

```
>>> a, b, c = 1, 2, 3
```

- 変数a とb の値を入れ替える素朴な方法

```
>>> temp = a
>>> a = b
>>> b = temp
```

タプルを使えば、1行で書ける！

```
>>> a, b = b, a
```

- 少し高度な技(?)の例

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

タプルによる戻り値

- 関数で複数の値を返したいときにタプルが使える
- 例：組み込み関数 divmod（商と剰余を返す）

```
>>> divmod(7,3)
(2, 1)  #商と剰余をタプルで返す
>>> quot, rem = divmod(7, 3)
```

- 数値リスト t の最大値と最小値を返す関数

```
def min_max(t):
    return min(t), max(t)
```

(発展)可変長引数

- *で始まるパラメータ名で、複数の引数をタプルにまとめることができる

```
def printall(*args):
    print(args)
>>> printall(1, 2.0, '3')
(1, 2.0, '3')
```

- 逆に、*タプル名で、タプルを展開して関数の引数に渡すことができる

```
>>> t = (7, 3)
>>> divmod(*t)
```

（まとめ）リストとタプル

- リスト(list)は変更可能なシーケンス

```
>>> xlst = [1, ‘abc’, 3.14] # 角括弧で作成
>>> xlst.append('xyz') #要素の追加
>>> xlst[0] = -10 #要素の変更
```

- 一方、タプル(tuple)は変更不可

```
>>> xtup = ('Taro',25,'Student') # 丸括弧で作成
```

  - 要素の変更や追加はNG
- 実際の使い分けの判断はなかなか難しい。。
  - リストの方が柔軟性は高い
  - タプルは辞書型のキーとして使える

zip関数によるタプルのリスト

- zip関数：複数の列(リスト又は文字列)を、タプルのリストに変換

```
>>> cities = ['Tokyo','Nagoya','Osaka','Sapporo']  #都市リスト
>>> temps = [25,26,28,21] #気温のリスト
>>> zip(cities,temps)
[('Tokyo', 25), ('Nagoya', 26), ('Osaka', 28), ('Sapporo', 21)]
```

- for 文と組み合わせると便利な場合もある

```
>>> for c,t in zip(cities,temps):
        print(c,':',t,'degrees')
Tokyo : 25 degrees
Nagoya : 26 degrees
Osaka : 28 degrees
Sapporo : 21 degrees
```

(発展)タプルのリストの並び替え

- 都市名と気温の組(タプル)のリスト

```
>>> citytemps = list(zip(cities,temps))
>>> print(citytemps)
[('Tokyo', 25), ('Nagoya', 26), ('Osaka', 28), ('Sapporo', 21)]
```

- 普通にソートすると

```
>>> citytemps.sort()
>>> print(citytemps)
[('Nagoya', 26), ('Osaka', 28), ('Sapporo', 21), ('Tokyo', 25)]
```

- 気温の低い順で並び替えるには？

```
>>> citytemps.sort(key=lambda a: a[1])
>>> print(citytemps)
[('Sapporo', 21), ('Tokyo', 25), ('Nagoya', 26),
('Osaka', 28)]
```

（参考）lambda式による無名関数

- 通常、関数はdef式を使って定義するが、lambda式を使って、「その場で」無名関数を作れると便利な場合もある
- 例：単語のリストを単語の長さ順でソートする

```
>>> greetings = ['hello','goodbye','bye','hi']
>>> greetings.sort(key=lambda x: len(x))
>>> greetings
['hi', 'bye', 'hello', 'goodbye']
```

辞書とタプル

- 辞書のitemsメソッド:含まれる キーと値のペアをタプルのシーケンスで返す

```
>>> d = {'a':0,'b':1,'c':2,'d':3}
>>> d.items()
dict_items([( ‘a’ , 0), ( ‘c’ , 2), ( ‘b’ , 1),
( ‘d’ , 3)])
# ただし順不同
```

- 例えば、値の大きい順でソートしたい場合は

```
>>> sorted(d.items(),key=lambda x:-x[1])
[('d', 3), ('c', 2), ('b', 1), ('a', 0)]
```

- タプルのリストから辞書を作成

```
>>> tempdict = dict(citytemps)
>>> print(tempdict)
{'Osaka': 28, 'Sapporo': 21, 'Nagoya': 26, 'Tokyo':
25}
```

今日の課題

英単語リスト words.txt (*) を読み込んで、以下を調べるプログラムを作成して提出せよ。

1. 文字数の多い単語、上位10個は？
2. 何文字の単語が最も多いか？
3. 各アルファベット文字（a-z）の出現回数は？
4. 同じ文字を2個以上含まない単語の中で最も文字数の多いのは？

- プログラムファイルに homework04.py という名前を付けて、ITC-LMSから提出すること
- 締切：6月19日午前8時（次回授業の直前）

(*) http://thinkpython.com/code/words.txt からダウンロード可。

課題の補足

ファイル words.txt から、全ての単語のリストを読み込む方法:

方法1（正直？なやり方）

```
>>> words = []
>>> fin = open('words.txt')
>>> for line in fin:
        words.append(line.strip())
>>> fin.close()
```

方法2（1行で済ませる方法）

```
>>> words = [line.strip() for line in open('words.txt').readlines()]
```

または、

```
>>> words = list(map(lambda a:a.strip(), open('words.txt').readlines()))
```

なお、ファイル入出力については、次週学ぶ予定。