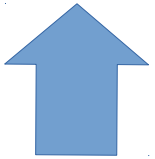
みんなのPython
07 華麗で短いプログラミング

山木翔馬

リスト内包表記

- 二乗を含むリスト

```
sq = [ x ** 2 for x in range( 10 ) ]
```



```
sq = [ ]
for x in range( 10 ) :
    sq.append( x ** 2 )
```

=> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

リスト内包表記での「if」

- 約数をリストに返す

```
val = 10
[ x for x in range( 1, val ) if val % x == 0 ]
```

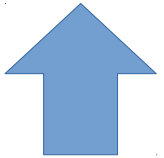
=> [1, 2, 5]

- ファイルの空白行を数える

```
lines = open( 'test.txt' )
len( [ l for l in lines if l.strip( ) == '' ] )
```

複数のfor文を記述したリスト内包表記

```
l = [ x+y for x in ( 1, 2, 3 ) for y in ( 100, 200, 300 ) ]
```



```
l = [ ]
for x in ( 1, 2, 3 ) :
    for y in ( 100, 200, 300 ) :
        l.append( x+y )
```

=> [101, 201, 301, 102, 202, 302, 103, 203, 303]

ディクショナリ内包表記
set内包表記

- 使い方はリスト内包表記と同じ
- ディクショナリ内包表記
  - d = { key:value for key, value in ... }
- set内包表記
  - s = { x for x in ... }

先頭のみ大文字にして重複しない名前のsetを作る

```
names = [ 'BOB', 'Burton', 'dave', 'bob' ]
unames = { x.titles( ) for x in names }
```

=> { 'Burton', 'Dave', 'Bob' }

イテレータ

- 次の要素を取り出す処理
- 要素が終わったかどうかを判別する処理

この2種類の処理を使って、複数の要素を持ったデータの要素を順番に取り出す

組み込み型とイテレータ

- 組み込み型をイテレータオブジェクトに変換

```
I = iter( 組み込み型 )
```

- next( )関数で次の要素を取り出す

```
>>> i = iter( [ 1, 2, 3 ] )
>>> next( i )
1
>>> next( i )
2
>>> next( i )
3
>>> next( i )   #=> StopIteration
```

イテレータの利点と欠点

利点

- サイズの大きいデータや、要素を取り出すのに時間がかかり処理に対しては、イテレータを使ったほうが効率的

欠点

- データのうち全ての要素を要求するような処理（要素数を数える処理など）では効率的でない

ジェネレータ

- ジェネレータはイテレータを簡潔に作成するためのツール
- 通常の関数のように定義するが、returnの代わりにyieldを使う

ジェネレータ関数の定義

```
def get_prime( x=2 ) :
    while True :
        for i in range( 2, x ) :
            if x % i == 0 :
                break
        else :
            yield x
        x += 1
```

```
>>> i = get_prime( )
>>> next( i )
2
>>> next( i )
3
>>> next( i )
5
```

ジェネレータ式

リスト内包表記と似た記述でジェネレータを作れる

```
>>> gen = ( ord( s ) for s in “Python” )
>>> next( gen )
80
>>> next( gen )
121
```

練習問題

- フィボナッチ数を生成するジェネレータ関数を定義せよ

```
>>> i = fib()
>>> for c in range(10):
...     print next(i),
...
0 1 1 2 3 5 8 13 21 34
```