파이썬 프로그래밍

약한 참조, 반복자, 발생자

한국기술교육대학교
온라인평생교육원

▣ 반복자

1. 반복자 객체

- 반복자 객체
  - next() 메소드를 지니고 있는 객체
  - next() 메소드로 더 이상 자료를 넘겨줄 수 없을 때 StopIteration 예외가 발생한다.
- 반복자 객체 생성 방법
  - iter(o) 내장 함수
  - 객체 o의 반복자 객체를 반환한다.
- 반복자 객체의 효율성
  - 반복자가 원 객체의 원소들을 복사하여 지니고 있지 않다.

```
I = iter([1,2,3])
print I

print I.next()
print I.next()
print I.next()
print I.next()
```

```
<listiterator object at 0x102648cd0>
1
2
3
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-20-98af67aeb8bc> in <module>()
      5 print I.next()
      6 print I.next()
----> 7 print I.next()

StopIteration:
```

▪iter()함수에서 반환되어지는 I가 반복자 객체

▪첫번째 I next() →1, 두 번째 →2, 세 번째 →3

▪마지막 I.next()는 가져올 수 있는 원소가 없어서 StopIteration 호출

▪I는 각 리스트의 원소를 뽑아낼 수 있는 기능만 있고 카피한 것은 X

▣ 반복자

1. 반복자 객체

- 리스트 객체에 대해 일반적인 for ~ in 반복 문 사용예

```
def f(x):
    print x + 1

for x in [1,2,3]:
    f(x)
```

```
2
3
4
```

- 리스트 객체에 반복자를 활용한 예

```
def f(x):
    print x + 1

t = iter([1,2,3])
while 1:
    try:
        x = t.next()
    except StopIteration:
        break
    f(x)
```

```
2
3
4
```

▪반복자 객체는 next()라는 메소드 가짐

▣ 반복자

1. 반복자 객체

- for ~ in 구문에 반복자를 활용할 수 있다.
  - for 문이 돌때 마다 반복자 객체의 next() 함수가 자동으로 호출되어 순차적으로 각 객체에 접근 가능하다.
  - StopIteration이 발생하면 for ~ in 구문이 멈춘다.

```
def f(x):
    print x + 1

t = iter([1,2,3])
for x in t:
    f(x)
```

```
2
3
4
```

▪반복자 객체를 for~in 구문의 in 뒤에 바로 사용 가능

▪t가 가지고 있는 next() 메소드도 자동으로 호출

▣ 반복자

1. 반복자 객체

```
def f(x):
    print x + 1

for x in iter([1,2,3]):
    f(x)
```

```
2
3
4
```

```
def f(x):
     print x + 1

for x in iter((1,2,3)):
     f(x)
```

```
2
3
4
```

▪반복자 객체를 정의하지 않고 바로 iter를 이용하여 직접 사용도 가능

▪반복자를 배우는 이유 : 발생자를 배우기 위한 것

▪iter의 객체로 리스트, 튜플 가능

▣ 반복자

2. 클래스에 반복자 구현하기

- 내장 함수 iter(o)에 대응되는 __iter__(self)의 구현
  - 객체 o에 iter(o)를 호출하면 자동으로 __iter__(self) 함수 호출
  - __iter__(self) 함수는 next() 함수를 지닌 반복자 객체를 반환해야 한다.

▣ 반복자

2. 클래스에 반복자 구현하기

```
class Seq:
    def __init__(self, fname):
        self.file = open(fname)
    #def __getitem__(self, n):
    #    if n == 10:
    #        raise StopIteration
    #    return n
    def __iter__(self):
        return self
    def next(self):
        line = self.file.readline() # 한 라인을 읽는다.
        if not line:
            raise StopIteration     # 읽을 수 없으면 예외 발생
        return line                 # 읽은 라인을 리턴한다.

s = Seq('readme.txt')     # s 인스턴스가 next() 메소드를 지니고 있으므로 s 인스턴스
                            자체가 반복자임
for line in s: # 우선 __iter__() 메소드를 호출하여 반복자를 얻고, 반복자에 대해서
                for ~ in 구문에 의하여 next() 메소드가 호출됨
    print line,

print

print Seq('readme.txt')

print list(Seq('readme.txt'))  # list() 내장 함수가 객체를 인수로 받으면 해당 객체의
                                반복자를 얻어와 next()를 매번 호출하여 각 원소를 얻어온다.
print tuple(Seq('readme.txt')) # tuple() 내장 함수가 객체를 인수로 받으면 해당 객체의
                                반복자를 얻어와 next()를 매번 호출하여 각 원소를 얻어온다.
```

▣ 반복자

2. 클래스에 반복자 구현하기

```
abc
def
ghi

<__main__.Seq instance at 0x10ddc5680>
['abc \ n', 'def \ n', 'ghi \ n']
('abc \ n', 'def \ n', 'ghi \ n')
```

- __iter__ →iter 내장함수와 매칭됨
- iter내장함수는 반복자 객체(자기 자신) 를 리턴
- self가 next()를 가지고 있음
- s 객체는 사용자가 생성한 클래스의 객체
- s에 __iter__이 존재해야 함 →없으면 error 발생
- for~in 구문에 s가 들어가면 iter 내장함수 호출됨
- self는 next() 메소드 호출
- for~in 구문에 iter 함수를 적지 않고 바로 리스트 적어도 무관
- 내장함수임과 동시에 리스트 클래스의 생성자
- 일단 iter를 불러준 뒤 반복자를 가져옴 →next 메소드 호출

▣ 반복자

3. 사전의 반복자

- 사전에 대해 for ~ in 구문은 키에 대해 반복한다.

```
d = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5}
for key in d:
    print key, d[key]
```

```
four 4
three 3
five 5
two 2
one 1
```

▪d에 대해서 먼저 iter 함수가 불림

```
d = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5}
for key in iter(d):
    print key, d[key]
```

```
four 4
three 3
five 5
two 2
one 1
```

▪반복자는 next를 가짐

▪next() →각각의 아이템의 key 값만 돌려줌

▪for~in에 쓰여지는 객체는 반드시 반복자 객체를 가짐

▣ 반복자

3. 사전의 반복자

• d.iterkeys() 함수
- 사전 d가 지닌 키에 대한 반복자 객체를 반환한다.

```
for key in d.iterkeys():   # 키에 대한 반복자, d.iterkeys() 가 반환한 반복자에 대해
                             next() 함수가 순차적으로 불리워짐
    print key,
```

```
four three five two one
```

▪iterkeys() →key 값만 가져옴
▪itervalues() →value 값만 가져옴

```
keyset = d.iterkeys()
print keyset.next()     # 반복자 객체는 항상 next() 메소드를 지니고 있음
for key in keyset:      # keyset 반복자에 대해 next() 메소드가 순차적으로 호출됨
    print key,
```

```
four
three five two one
```

▪iterkeys() →key 값만 가져와서 반복하는 반복자
▪next가 한 번 호출되면 반복자는 그 다음 next는 다음 원소 활용

▣ 반복자

3. 사전의 반복자

• d.itervalues() 함수
- 사전 d가 지닌 값에 대한 반복자 객체를 반환한다.

```
for value in d.itervalues():    # 값에 대한 반복자
    print value,
```

```
4 3 5 2 1
```

```
for key, value in d.iteritems(): #(키,값)에 대한 반복자
    print key, value
```

```
four 4
three 3
five 5
two 2
one 1
```

▣ 반복자

4. 파일 객체의 반복자

• 파일 객체는 그 자체가 반복자임
- next() 함수에 의해 각 라인이 순차적으로 읽혀짐

```
f = open('readme.txt')
print "f.next()", f.next()
for line in f:   # f.next() 가 순차적으로 호출됨
    print line,
```

```
f.next() 1: Hello World

2: Hello World
3: Hello World
4: Hello World
5: Hello World
```

▪f 파일 객체 자체가 반복자 →next를 가지고 있기 때문
▪콤마(,)가 있어도 옆에 같이 출력 안되는 이유는 개행 문자 때문