이터레이터와 제너레이터

장혜식

Python Software Foundation

LinuxKorea Inc.

이터레이터 도입

- 이터레이터 *PEP* 제출 *(2001*년 *1*월, *Ka-Ping Yee, GvR)*
- 파이썬 *2.2*에 *__future__*로 적용, 릴리즈 *(2001*년 *12*월*)*
- 파이썬 *2.3*에 정규 적용, 릴리즈 *(2003*년 *7*월*)*

이터레이터 도입 목적

- *for*루프 등 반복적으로 다음값을 가져오는 작업을 구조화
- 리스트 값을 이터레이트시 성능 향상
- 딕셔너리 값을 이터레이트시 "엄청난" 성능 향상
- 시퀀스가 아닌 것도 이터레이트 가능

이터레이터 전과 후 (1)

전

```
d = {}
for k, v in d.items():
    print k, v
```

후

```
d = {}
for k, v in d.iteritems():
    print k, v
```

이터레이터 전과 후 (2)

전

```
d = {}
if d.has_key('xyz'):
    do_something()
```

후

```
d = {}
if 'xyz' in d:
   do_something()
```

이터레이터 전과 후 (3)

전

```
f = open("file")
while True:
    line = f.readline()
    if not line:
        break
    do_something()
```

후

```
for line in open("file"):
    do_something()
```

이터레이터 전과 후 (4)

전

```
a, b = [], []
acur, bcur = 0, 0
while True:
    if some():
        v = a[acur]
        acur += 1
    else:
        v = b[bcur]
        bcur += 1
    do_something()
```

후

```
a, b = [], []
ai, bi = iter(a), iter(b)
while True:
    if some(): v = ai.next()
    else: v = bi.next()
    do_something()
```

이터레이터 versus 리스트

- 이터레이터는 한꺼번에 일어나는 메모리 할당이 적기때문에, 큰 리스트를 만들어야 하는 상황에 적합.
- 값을 뽑아내며 진행한다는 뜻이 명확하기 때문에, 앞으로 계속 많은 부분이 이터레이터로 전환 예정.

이터레이터 사용

- *iter(o)* => *o*의 이터레이터 리턴
- *i.next()* => 다음 값 리턴 *(*끝나면 *StopIteration* 예외 발생*)*
- *iter(func, sentinel)* => 콜 이터레이터

이터레이터 객체 정의 (1)

```
class ICanBeIterated(object):
    def __iter__(self):
        return iter([1,2,3])
```

이터레이터 객체 정의 (2)

```
import random

class ICanBeIterated(object):
    def __iter__(self):
        return self
    def next(self):
        return random.randrange(100)
```

itertools 모듈의 목적

- *for, while* 등의 루프들의 자주 반복되는 패턴들을 간단하고 보기 좋고 빠르게 사용.
- 같은 오브젝트를 반복*(repeat)*, 여러 이터레이션을 연결*(chain)*, 숫자만 세기*(count)* 등 여러가지 유용한 함수.
- *zip, filter* 등 기존 빌트인의 이터레이터 버전도 포함.

itertools 모듈 (1)

- *chain(it, it..):* 여러 이터레이터를 연결
- *count(n=0): n*부터 숫자를 셈
- *repeat(it, times):* 한 오브젝트를 여러 번 반복
- *cycle(it):* 같은 이터레이션을 여러 번 반복

itertools 모듈 (2)

- *dropwhile(pred, it):* 이터레이션을 하면서 pred(i)가 참일 때까지 무시.
- *takewhile(pred, it): pred(i)*가 참일 때까지만 이터레이트.

itertools 모듈 (3)

- *ifilter(pred, it):* 빌트인 *filter* 함수
- *ifilterfalse(pred, it): ifilter*와 반대로 거짓만 취함
- *imap(func, *it):* 빌트인 *map* 함수
- *islice(iter, [start,] stop [, step]):* 시퀀스에서 범위, 스텝을 조절해서 이터레이션
- *izip(*it):* 빌트인 *zip* 함수

itertools 모듈 (4)

- *2.4*에서 추가된 이터레이터들
- *groupby(it[, key]): key*함수의 리턴값을 기준으로 그룹 이터레이터를 내부에서 다시 이터레이트. *(SQL*의 *GROUPBY)*
- *tee(it):* 이터레이터를 여러 군데에서 쓸 수 있도록 분리. *(*유닉스의 *tee(1))*

2.4의 새로운 보조 도구

- *reversed(it):* 이터레이터를 뒤집음.
- *sorted(list, cmp=None, key=None, reverse=False):* 리스트또는 이터레이터를 정렬해서 리턴 (리스트).
- *operator.itemgetter(item): lambda x:x[item]*
- *operator.attrgetter(item): lambda x:x.item*

제너레이터 도입

- *"Simple Generators" PEP제출 (2001년 5월, Neil S., Tim Peters)*
- 파이썬 *2.2*에 *__future__*로 적용, 릴리즈 *(2001년 12월)*
- 파이썬 *2.3*에 정규 적용, 릴리즈 *(2003년 7월)*
- *PEP289 "Generator Expression"* 제출 *(2002년 1월, Raymond Hettinger)*
- 제너레이터 익스프레션 구현 제출 *(2004년 1월, 서지원)*

Simple Generator 문법

```
def fibgen():
    a, b = 0, 1
    while True:
        yield b
        a, b = b, a+b
```

심플 제너레이터의 장점

- 다른 루틴에 값을 지속적으로 전달하면서도, 실행 문맥이 끊기지 않는다.
- 동시에 여러 루틴이 실행될 수 있는 잠재적인 활용 요소. (멀티플렉싱)
- 함수나 클래스 메쏘드에 비해 재진입 속도가 빠름.
- 매우 쉽다.

심플 제너레이터 활용 (1)

- 간단한 ini 파일을 읽는 제너레이터

심플 제너레이터 활용 (2)

- 각 채널별로 제너레이터를 할당한 예제

제너레이터 익스프레션

- *list comprehension*의 성공 이후 쇄도하는 터플, 딕셔너리, 이터레이터 등 다른 타입 요청을 통합하기 위해서.
- *[]* 대신 *()*로 묶어서 한 줄로 제너레이터를 만듦.
- 차후에는 *list comprehension*을 완전히 대체할 예정.

제너레이터 익스프레션 문법

```
(x for x in ‘hello’)
((x, y) for x in ‘hello’
        for y in ‘123’)
(x+y for x in range(10)
     for y in range(11, 20)
     if x*y % 3 == 0)
list(x for x in ‘hello’)
```

list comp와 다른 점

- 값이 동시에 생성되지 않고, 매 이터레이션에 생성
- *in 1,2,3]* 형태가 허용되지 않음.

바인딩 시점 이슈

- 얼리 바인딩 *(Early-binding):* 정의 시점에 변수들을 모두 잡아와서, 실행되는 순간의 상태의 변수들이 영향을 미치지 않음. *lambda*나 *listcomp*와 호환되지 않는 문제점이 있음.
- 레이지 바인딩 *(Lazy-binding):* 기존의 *lambda*와 같이 실행 시점의 네임 스페이스를 기준으로 함. 코드는 문제 소지가 있을 수 있지만 일관성을 얻을 수 있음.

Generator Expression 전망

- 현재는 소스포지 패치에 등록된 상태.
- *GvR*은 *2.4alpha1* 이전에 *lazy-binding*으로 일단 적용하고 그 이후 실 사용자들의 의견을 수렴하여 결정하기로 제안.
- 서지원님이 퇴소한 뒤에 파이널 리뷰와 트렁크 커밋 예정.
- *Python 3*에서는 *listcomp*와 통합.

이터레이터와 제너레이터

끝