파이썬 프로그래밍

약한 참조, 반복자, 발생자

한국기술교육대학교
온라인평생교육원

▣ 발생자

1. 발생자란?

- 발생자(Generator)
  - (중단된 시점부터) 재실행 가능한 함수
- 아래 함수 f()는 자신의 인수 및 내부 변수로서 a, b, c, d를 지니고 있다.
  - 이러한 a, b, c, d 변수들은 함수가 종료되고 반환될 때 모두 사라진다.
- 발생자는 f()와 같이 함수가 종료될 때 메모리에서 해제되는 것을 막고 다시 함수가 호출 될 때 이전에 수행이 종료되었던 지점 부터 계속 수행이 가능하도록 구현된 함수이다.

```
def f(a,b):
    c = a * b
    d = a + b
    return c, d
```

▪발생자는 함수

▪f 함수는 리턴이 되면 a,b,c,d 모두 사라짐

▣ 발생자

1. 발생자란?

- yield 키워드
  - return 대신에 yield에 의해 값을 반환하는 함수는 발생자이다.
  - yield는 return과 유사하게 임의의 값을 반환하지만 함수의 실행 상태를 보존하면서 함수를 호출한 쪽으로 복귀시켜준다.
- 발생자는 곧 반복자이다.
  - 즉, 발생자에게 next() 호출이 가능하다.

```
def generate_ints(N):
    for i in range(N):
        yield i
```

▪발생자를 만들려면 return 대신 yield 키워스 사용

▣ 발생자

1. 발생자란?

```
gen = generate_ints(3) # 발생자 객체를 얻는다. generate_ints() 함수에 대한 초기
                         스택 프레임이 만들어지나 실행은 중단되어 있는 상태임
print gen
print gen.next() # 발생자 객체는 반복자 인터페이스를 가진다. 발생자의 실행이 재개됨.
                   yield에 의해 값 반환 후 다시 실행이 중단됨
print gen.next() # 발생자 실행 재개. yield에 의해 값 반환 후 다시 중단
print gen.next() # 발생자 실행 재개. yield에 의해 값 반환 후 다시 중단
print gen.next() # 발생자 실행 재개. yield에 의해 더 이상 반환할 값이 없다면
                   StopIteration 예외를 던짐
```

```
<generator object generate_ints at 0x10ddd6410>
0
1
2
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-8-65149ac25109> in <module>()
      4 print gen.next() # 발생자 실행 재개. yield에 의해 값 반환 후 다시 중단
      5 print gen.next() # 발생자 실행 재개. yield에 의해 값 반환 후 다시 중단
----> 6 print gen.next() # 발생자 실행 재개. yield에 의해 더 이상 반환할 값이 없다면
                          StopIteration 예외를 던짐

StopIteration:
```

- __generate init__ →발생자 함수
- gen →yield가 반환하는 I 값을 가짐
- 기본적으로는 gen에는 __generate init__ 자체의 객체가 들어감
- 발생자가 곧 반복자로 발생자 gen에게 next 호출 가능
- 마지막 next는 돌려줄 값이 없으므로 StopIteration 발생

▣ 발생자

1. 발생자란?

- 위와 같은 세부 동작 방식을 이용하여, 다음과 같이 for ~ in 구문에 적용할 수 있다.

```
for i in generate_ints(5):
    print i,
```

```
0 1 2 3 4
```

▪for~in 구문에 발생자 바로 사용 가능
▪발생자는 곧 반복자 →next를 가지고 있음
▪예외가 발생하면 for~in 구문은 빠져나가게 되어 있음

- 발생자 함수와 일반 함수의 차이점
  - 일반 함수는 함수가 호출되면 그 함수 내부에 정의된 모든 일을 마치고 결과를 반환함
  - 발생자 함수는 함수 내에서 수행 중에 중간 결과 값을 반환할 수 있음
- 발생자가 유용하게 사용되는 경우
  - 함수 처리의 중간 결과를 다른 코드에서 참조할 경우
  - 모든 결과를 한꺼번에 처리하는 것이 아니라 함수 처리 중에 나온 중간 결과를 사용해야 할 경우

▣ 발생자

2. 발생자 구문

- 리스트 내포(List Comprehension)
  - 리스트 객체의 새로운 생성
  - 메모리를 실제로 점유하면서 생성됨

```
print [k for k in range(100) if k % 5 == 0]
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
```

▪k가 리스트의 원소

▣ 발생자

2. 발생자 구문

- 리스트 내포 구문에 []가 아니라 () 사용
  - 리스트 대신에 발생자 생성
  - 처음부터 모든 원소가 생성되지 않고 필요한 시점에 각 원소가 만들어짐
  - 메모리를 보다 효율적으로 사용함

```
a = (k for k in range(100) if k % 5 == 0)
print a
print a.next()
print a.next()
print a.next()
for i in a:
    print i,
```

```
<generator object <genexpr> at 0x10d84df50>
0
5
10
15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95
```

▪리스트 내포와 문법이 똑같은데, []가 아닌 () 사용 →발생자

▪리스트 내포 : 리스트 전반적인 내용 모두 생성

▪발생자 : next가 호출될 때마다 리스트 내용 발생

▣ 발생자

2. 발생자 구문

- 아래 예는 sum 내장 함수에 발생자를 넣어줌
  - sum을 호출하는 시점에는 발생자가 아직 호출되기 직전이므로 각 원소들은 아직 존재하지 않는다.
  - sum 내부에서 발생자가 지니고 있는 next() 함수를 호출하여 각 원소들을 직접 만들어 활용한다.
  - 메모시 사용 효율이 높다.

```
print sum((k for k in range(100) if k % 5 == 0))
```

```
950
```

▪sum을 호출하는데 안쪽 인자가 발생자 k를 generate

▣ 발생자

3. 발생자의 활용 예 1 - 피보나치 수열

```
def fibonacci(a = 1, b = 1):
    while 1:
        yield a
        a, b = b, a + b

for k in fibonacci():  # 발생자를 직접 for ~ in 구문에 활용
    if k > 100:
        break
    print k,
```

```
1 1 2 3 5 8 13 21 34 55 89
```

▪yield가 있으니까 fibonaci는 발생자

▣ 발생자

4. 발생자의 활용 예 2 - 홀수 집합 만들기

• 반복자를 활용한 예

```
class Odds:
    def __init__(self, limit = None): # 생성자 정의
        self.data = -1                # 초기 값
        self.limit = limit            # 한계 값
    def __iter__(self):               # Odds 객체의 반복자를 반환하는 특수 함수
        return self
    def next(self):                   # 반복자의 필수 함수
        self.data += 2
        if self.limit and self.limit <= self.data:
            raise StopIteration
        return self.data

for k in Odds(20):
    print k,
print
print list(Odds(20)) # list() 내장 함수가 객체를 인수로 받으면 해당 객체의 반복자를
                     얻어와 next()를 매번 호출하여 각 원소를 얻어온다.
```

```
1 3 5 7 9 11 13 15 17 19
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

▪Odds(20) →Odds라는 클래스에는 인스턴스 메소드로 if를 가짐

▪반복자를 next 를 가지고 있는 self로 가져옴

▣ 발생자

4. 발생자의 활용 예 2 - 홀수 집합 만들기

- 발생자를 활용한 예

```
def odds(limit=None):
    k = 1
    while not limit or limit >= k:
        yield k
        k += 2

for k in odds(20):
    print k,
print
print list(odds(20)) # list() 내장 함수가 발생자를 인수로 받으면 해당 발생자의 next()를
                     매번 호출하여 각 원소를 얻어온다.
```

```
1 3 5 7 9 11 13 15 17 19
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

▪for~in 구문에 odds(20)사용 가능 →odds라는 반복자는 곧 발생자

▪list도 역시 발생자를 받으면 반복자로 사용