12강

제너레이터

12-1 제너레이터 개요

- 제너레이터
  - ✓ 제너레이터는 이터레이터를 생성해주는 함수를 말한다.
  - ✓ 이터레이터는 클래스에 __iter__, __next__ 또는 __getitem__ 메서드를 구현해야 하지만 제너레이터는 함수 안에서 yield라는 키워드만 사용하면 된다.
  - ✓ 제너레이터는 이터레이터보다 훨씬 간단하게 작성할 수 있는 장점을 가지고 있다.
  - ✓ 제너레이터는 발생자라고도 부른다.

12-1 제너레이터 개요

- yield문
  - ✓ 함수 안에서 yield를 사용하면 함수는 제너레이터가 되며 yield에는 전달 할 값(변수)을 지정한다.
  - ✓ 이터레이터는 __next__ 메서드 안에서 직접 return으로 값을 반환했지만 제너레이터는 yield에 지정한 값이 __next__ 메서드(next 함수)의 반환값이 된다.

```
def number_generator():
    yield 0

    yield 1

    yield 2


for i in number_generator():
    print(i)
```

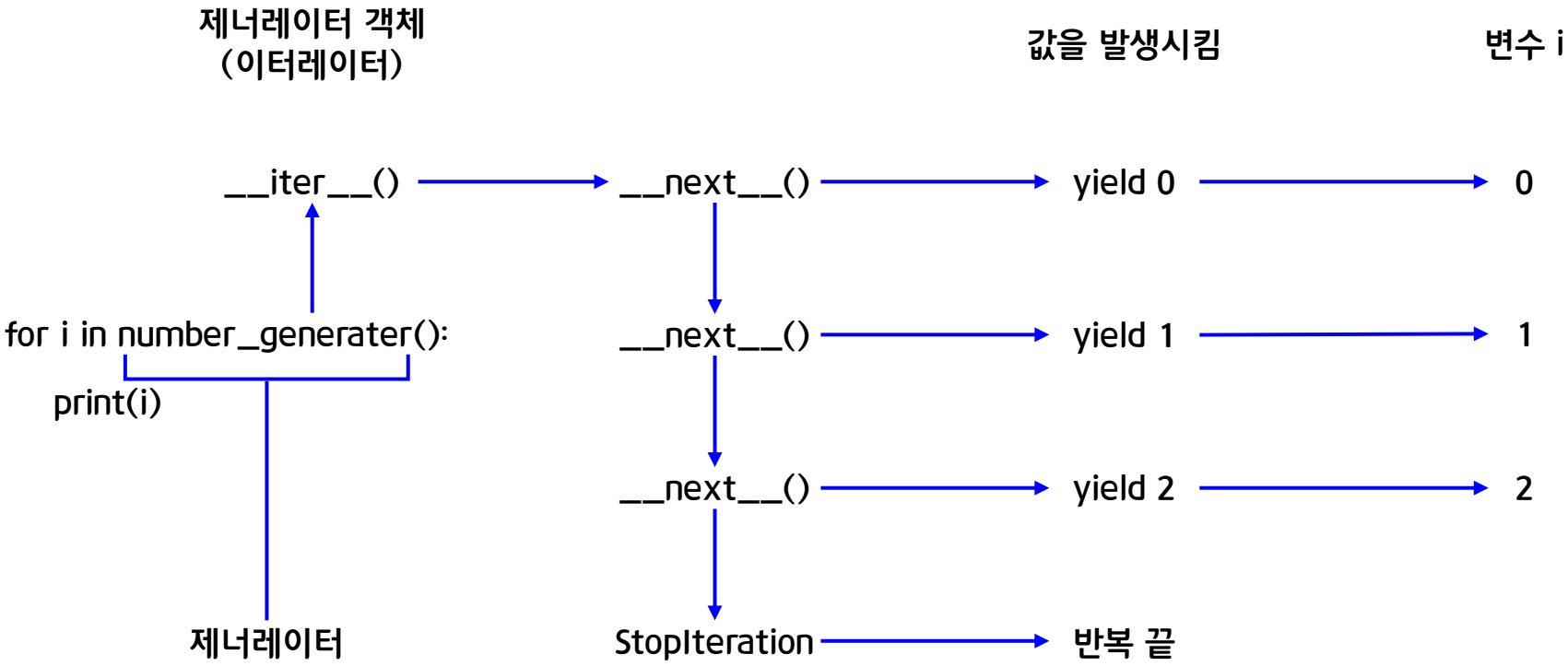
```
[결과]
0
1
2
```

```
g = number_generator()
print(g.__next__())
print(g.__next__())
print(g.__next__())
print(g.__next__())
```

```
[결과]
0
1
2
Traceback (most recent call last):
  File "C:/Users/generatorEx1.py", line 5, in <module>
    print(g.__next__())
StopIteration
```

12-1 제너레이터 개요

- 제너레이터와 for문
  - ✓ for 반복문은 반복할 때마다 __next__를 호출하여 yield에서 발생한 리턴 값을 가져온다.



12-2 제너레이터 생성

- 제너레이터 생성
  - ✓ 아래 코드처럼 제너레이터를 생성할 수 있다.

```
def CRange3(stop):
    n = 0
    while n < stop:
        yield n
        n+=1

for i in CRange3(3):
    print(i)
```

```
[결과]
0
1
2
```

```
def strdata_upper(datas):
    for str in datas:
        yield str.upper()

foods = ['juice', 'steak', 'bread', 'coffee']

for data in strdata_upper(foods):
    print(data)
```

```
[결과]
JUICE
STEAK
BREAD
COFFEE
```

12-2 제너레이터 생성

- yield from
  - ✓ 데이터가 여러 개인 경우 매번 반복문을 사용하지 않고, yield from을 사용하면 한 번에 데이터를 전송할 수 있다.

```
def mutiple_number_generator():
    data = [10,20,30,40]
    yield from data

for idata in mutiple_number_generator():
    print(idata)
```

```
[결과]
10
20
30
40
```

```
def n_generator(stop):
    n = 0
    while n < stop:
        yield n
        n+=1


def t_generator():
    yield from n_generator(3)

for i in t_generator():
    print(i)
```

```
[결과]
0
1
2
```