Unit 40.

Status	완료
담당자	
마감일	
완료일	@2022년 12월 2일

Unit 40 제너레이터 사용하기

제너레이터는 이터레이터를 생성해주는 함수

제너레이터는 함수 안에서 yield라는 키워드만 사용하면 끝

40.1 제너레이터와 yield 알아보기

함수 안에서 yield를 사용하면 함수는 제너레이터가 되며 yield에는 값(변수)을 지정한다.

• yield 값

```
def number_generator():
    yield 0
    yield 1
    yield 2

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

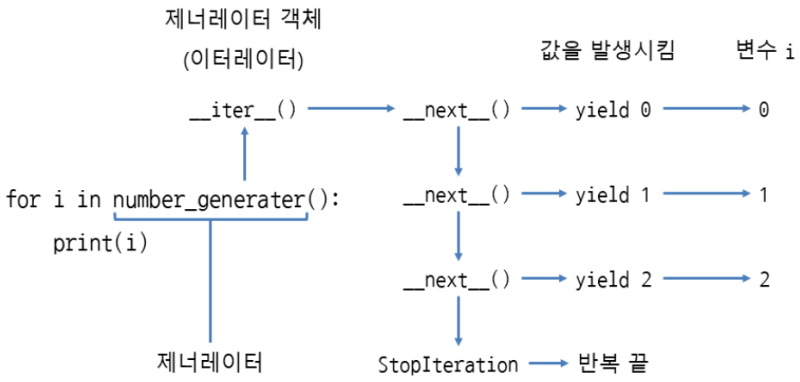
40.1.1 제너레이터 객체가 이터레이터인지 확인하기

이터레이터와 마찬가지로 dir 함수로 메서드 목록을 확인해서 **iter**, **next** 메서드가 있는지 확인한다

제너레이터는 제너레이터 객체에서 **next** 메서드를 호출할 때마다 함수 안의 yield까지 코드를 실행하며 yield에서 값을 발생시킨다

40.1.2 for와 제너레이터

for 반복문은 반복할 때마다 __next__를 호출하므로 yield에서 발생시킨 값을 가져온다.



40.1.3 yield의 동작 과정 알아보기

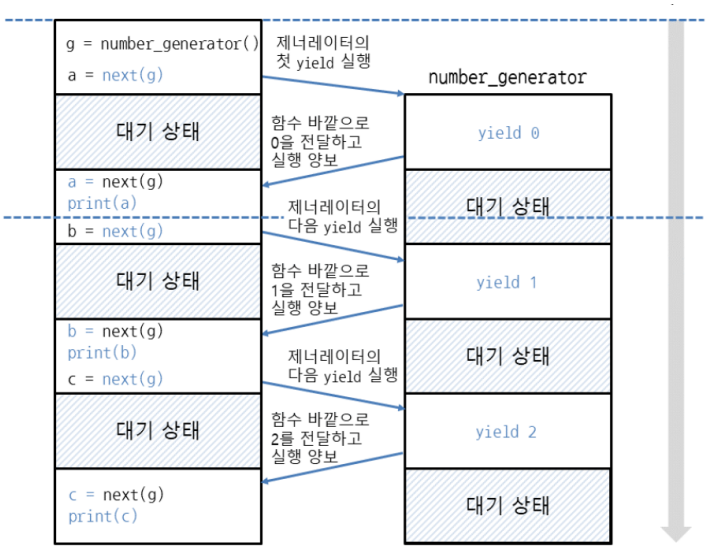
• 변수 = next(제너레이터객체)

```
def number_generator():
    yield 0    # 0을 함수 바깥으로 전달하면서 코드 실행을 함수 바깥에 양보
    yield 1    # 1을 함수 바깥으로 전달하면서 코드 실행을 함수 바깥에 양보
    yield 2    # 2를 함수 바깥으로 전달하면서 코드 실행을 함수 바깥에 양보

g = number_generator()

a = next(g)    # yield를 사용하여 함수 바깥으로 전달한 값은 next의 반환값으로 나옴
print(a)       # 0
```

```
b = next(g)
print(b)       # 1

c = next(g)
print(c)       # 2
```

yield를 사용하여 바깥으로 전달한 값은 next 함수의 반환값으로 나온다.

제너레이터 함수가 실행되는 중간에 next로 값을 가져온다.



yield 1이 실행되고 숫자 1을 발생시켜서 바깥으로 전달하고 함수 바깥에서는 print(b)로 next(g)에서 반환된 값을 출력

40.2 제너레이터 만들기

range(횟수)처럼 동작을 하는 제너레이터

```
def number_generator(stop):
    n = 0              # 숫자는 0부터 시작
    while n < stop:    # 현재 숫자가 반복을 끝낼 숫자보다 작을 때 반복
        yield n        # 현재 숫자를 바깥으로 전달
        n += 1         # 현재 숫자를 증가시킴

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

40.2.1 yield에서 함수를 호출하기

yield i.upper()와 같이 yield에서 함수(메서드)를 호출하면 해당 함수의 반환값을 바깥으로 전달

upper는 호출했을 때 대문자로 된 문자열을 반환

```
def upper_generator(x):
    for i in x:
        yield i.upper()    # 함수의 반환값을 바깥으로 전달

fruits = ['apple', 'pear', 'grape', 'pineapple', 'orange']
for i in upper_generator(fruits):
    print(i)
```

40.3 yield from으로 값을 여러 번 바깥으로 전달하기

값을 여러 번 바깥으로 전달할 때는 for 또는 while 반복문으로 반복하면서 yield를 사용

```
def number_generator():
    x = [1, 2, 3]
    for i in x:
```

```
        yield i

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

이런 경우에는 매번 반복문을 사용하지 않고, yield from을 사용하면 된다.

yield from에는 반복 가능한 객체, 이터레이터, 제너레이터 객체를 지정

- **yield from 반복가능한객체**
- **yield from 이터레이터**
- **yield from 제너레이터객체**

```
def number_generator():
    x = [1, 2, 3]
    yield from x    # 리스트에 들어있는 요소를 한 개씩 바깥으로 전달

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

yield from을 한 번 사용하여 값을 세 번 바깥으로 전달한다.

40.3.1 yield from에 제너레이터 객체 지정하기

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

def three_generator():
    yield from number_generator(3)    # 숫자를 세 번 바깥으로 전달

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

제너레이터 number_generator는 매개변수로 받은 숫자 직전까지 숫자를 만들어낸다.