11강

이터레이터

11-1 이터레이터 개요

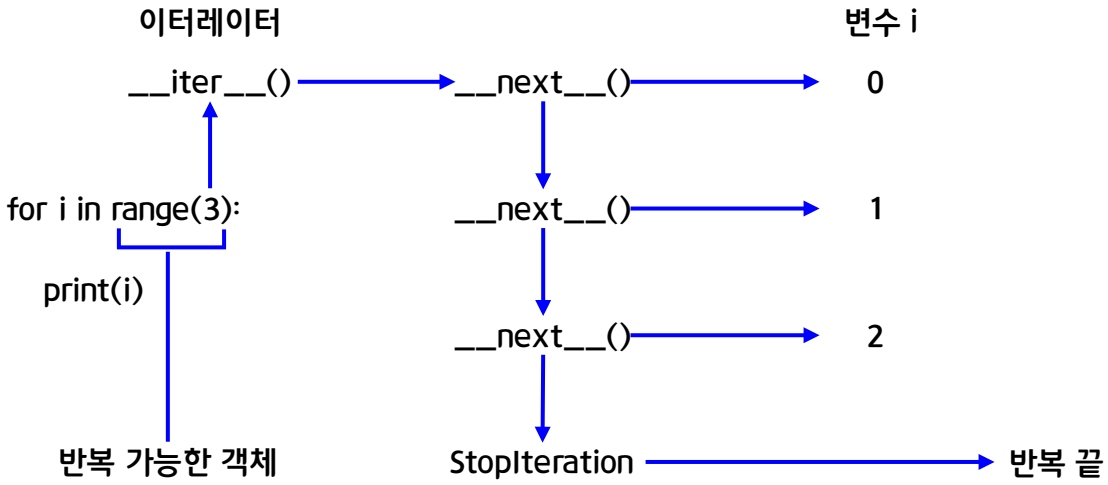
- 이터레이터
  - ✓ 이터레이터는 반복자라고 하는데 여러 개의 요소를 차례대로 꺼낼 수 있는 객체를 얘기한다.
  - ✓ 데이터 생성을 뒤로 미루는 방식을 지연 평가(lazy evaluation)라고 한다.
  - ✓ 파이썬에서는 이터레이터만 생성하고 데이터가 필요한 시점에 데이터를 생성하는 방식인 지연 평가 방식을 사용한다.

```
iterator = [1,2,3,4].__iter__()
iterator.__next__()
1
iterator.__next__()
2
iterator.__next__()
3
iterator.__next__()
4
iterator.__next__()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
```

```
iterator = range(4).__iter__()
iterator.__next__()
1
iterator.__next__()
2
iterator.__next__()
3
iterator.__next__()
4
iterator.__next__()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
```

11-1 이터레이터 개요

- for문 이터레이터
  - ✓ 아래와 같이 for에 range(3)을 사용하면 for문에서는 우선 range객체의 __iter__()를 호출하여 이터레이터 얻어 온다.
  - ✓ for 문에 의해 반복할 때마다 __iter__()로 부터 전해 받은 이터레이터 객체의 __next__()를 호출 해서 값을 전달 받아 i에 저장한다.
  - ✓ 지정된 숫자 3이 되면 __next__()은 값 대신 StopIteration을 발생시켜서 for문의 반복을 끝내도록 한다.



11-2 이터레이터 생성

- 이터레이터 생성
  - ✓ 이터레이터를 생성하기 위해서 클래스를 사용한다.
  - ✓ 이 클래스에는 __iter__()와 __next__()를 구현해야 한다.
  - ✓ __iter__()는 객체 주소를 넘겨준다.
  - ✓ __next__()는 for 문에 넘겨 줄 데이터를 정의하고 특정 범위가 넘어 갔을 때 for문을 멈추기 위해 StopIteration을 발생시킨다.

```
class CustomRange:
    def __init__(self,start, stop):
        self.current = start
        self.stop = stop

    def __iter__(self):
        return self

    def __next__(self):
        if self.current < self.stop:
            r = self.current
            self.current += 1
            return r
        else:
            raise StopIteration

for i in CustomRange(0,4):
    print(i)
```

```
[결과]
0
1
2
3
```

11-2 이터레이터 생성

- 이터레이터 언패킹
  - ✓ 아래와 같이 CustomRang()의 결과를 언패킹할 수 있다.
  - ✓ 이터레이터가 반복하는 횟수와 변수의 개수는 같아야 한다.
  - ✓ 앞에서 사용한 map도 이터레이터이다.
  - ✓ num1, num2 = map(int, input().split())처럼 언패킹으로 변수 여러 개에 값을 할당할 수 있다.

```
num1, num2, num3 = CustomRange(0,3)
print(num1,num2,num3)

num1, num2, num3, num4 = CustomRange(0,4)
print(num1,num2,num3, num4)
```

```
[결과]
0 1 2
0 1 2 3
```

11-2 이터레이터 생성

- 이터레이터 인덱싱
  - ✓ 인덱싱을 이용할 수 있는 이터레이터를 구성하기 위해서는 __getitem__()를 구현해야 한다.
  - ✓ __getitem__()에서는 인덱스 범위가 넘어서 값이 들어 올 때 IndexError를 발생시켜야 한다.

```
class CustomRange:
    def __init__(self, stop):
        self.current = 0
        self.stop = stop

    def __iter__(self):
        return self

    def __next__(self):
        if self.current < self.stop:
            r = self.current
            self.current += 1
            return r
        else:
            raise StopIteration

    def __getitem__(self, index):
        if index < self.stop:
            return index
        else:
            raise IndexError

print(CustomRange(3)[1])
print(CustomRange(3)[2])
```

```
[결과]
1
2
```