Unit 39.

Status	완료
담당자	
마감일	
완료일	@2022년 12월 1일

Unit 39. 이터레이터 사용하기

이터레이터는 값을 차례대로 꺼낼 수 있는 객체이다.

39.1 반복 가능한 객체 알아보기

문자열, 리스트, 딕셔너리, 세트가 반복 가능한 객체 요소가 여러 개 들어있고 한 번에 하나씩 꺼낼 수 있는 객체이다.

객체가 반복 가능한 객체인지 알아보는 방법은 객체에 _ iter _ 메서드가 들어있는지 확인

dir 함수를 사용하면 객체의 메서드를 확인할 수 있다

• dir(객체)

```
>>> dir([1, 2, 3])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__g
etattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__seta
ttr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'inser
t', 'pop', 'remove', 'reverse', 'sort']
```

이 리스트에서 _ iter _를 호출해보면 이터레이터로 나온다

```
>>> [1, 2, 3].__iter__()
<list_iterator object at 0x03616630>
```

리스트의 이터레이터를 변수에 저장한 뒤 **next** 메서드를 호출해보면 요소를 차례대로 꺼낼 수 있다.

이터레이터는 요소를 계속해서 꺼내다가 꺼낼 요소가 없으면 StopIteration 예외를 발생시켜서 반복을 끝낸다.

리스트뿐만 아니라 문자열, 딕셔너리, 세트도 같다(리스트, 문자열, 딕셔너리, 세트는 요소가 눈에 보이는 반복 가능한 객체).

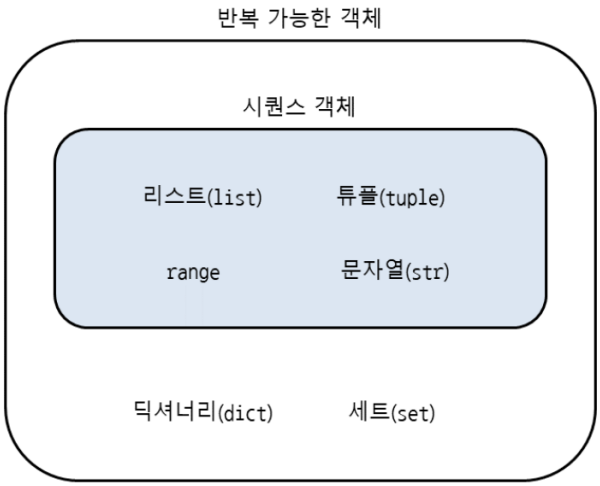
이번에는 요소가 눈에 보이지 않는 range를 살펴보자

```
>>> it = range(3).__iter__()
>>> it.__next__()
0
>>> it.__next__()
1
>>> it.__next__()
2
>>> it.__next__()
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    it.__next__()
StopIteration
```

range(3)이므로 0, 1, 2 세 번 반복하며 요소가 눈에 보이지 않지만 지정된 만큼 숫자를 꺼내서 반복할 수 있다.

39.1.1 for와 반복 가능한 객체

- 반복 가능한 객체는 요소를 한 번에 하나씩 가져올 수 있는 객체
- 이터레이터는 **next** 메서드를 사용해서 차례대로 값을 꺼낼 수 있는 객체



39.2 이터레이터 만들기

iter, **next** 메서드를 직접 구현해서 이터레이터 만들기

```
class 이터레이터이름:
    def __iter__(self):
        코드

    def __next__(self):
        코드
```

```
class Counter:
    def __init__(self, stop):
        self.current = 0    # 현재 숫자 유지, 0부터 지정된 숫자 직전까지 반복
        self.stop = stop    # 반복을 끝낼 숫자

    def __iter__(self):
        return self         # 현재 인스턴스를 반환

    def __next__(self):
        if self.current < self.stop:    # 현재 숫자가 반복을 끝낼 숫자보다 작을 때
            r = self.current            # 반환할 숫자를 변수에 저장
            self.current += 1           # 현재 숫자를 1 증가시킴
            return r                    # 숫자를 반환
        else:                           # 현재 숫자가 반복을 끝낼 숫자보다 크거나 같을 때
            raise StopIteration         # 예외 발생

for i in Counter(3):
    print(i, end=' ')
```

39.2.1 이터레이터 언패킹

Counter()의 결과를 변수 여러 개에 할당할 수 있다 .

이터레이터가 반복하는 횟수와 변수의 개수는 같아야 한다.

```
>>> a, b, c = Counter(3)
>>> print(a, b, c)
0 1 2
>>> a, b, c, d, e = Counter(5)
>>> print(a, b, c, d, e)
0 1 2 3 4
```

39.3 인덱스로 접근할 수 있는 이터레이터 만들기

getitem 메서드를 구현하여 인덱스로 접근할 수 있는 이터레이터를 만들어보겠다.

```
class 이터레이터이름:
    def __getitem__(self, 인덱스):
        코드
```

```
class Counter:
    def __init__(self, stop):
        self.stop = stop

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

print(Counter(3)[0], Counter(3)[1], Counter(3)[2])

for i in Counter(3):
    print(i, end=' ')
```

클래스에서 __getitem__만 구현해도 이터레이터가 되며 **iter**, __next__는 생략해도 된다.

여기서는 Counter(3)처럼 반복을 끝낼 숫자를 받았으므로 self.stop에 stop을 넣어준다.

```
class Counter:
    def __init__(self, stop):
        self.stop = stop             # 반복을 끝낼 숫자
```

39.4 iter, next 함수 활용하기

iter는 객체의 **iter** 메서드를 호출해주고, next는 객체의 **next** 메서드를 호출해준다.

```
>>> it = iter(range(3))
>>> next(it)
0
>>> next(it)
1
>>> next(it)
2
>>> next(it)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    next(it)
StopIteration
```

반복 가능한 객체에서 __iter__를 호출하고 이터레이터에서 **next** 메서드를 호출

이외에 다른 기능들도 있다.

39.4.1 iter

iter는 반복을 끝낼 값을 지정하면 특정 값이 나올 때 반복을 끝낸다. 이 경우에는 반복 가능한 객체 대신 호출 가능한 객체를 넣는다.

• iter(호출가능한객체, 반복을끝낼값)

```
>>> import random
>>> it = iter(lambda : random.randint(0, 5), 2)
>>> next(it)
0
>>> next(it)
3
>>> next(it)
1
>>> next(it)
Traceback (most recent call last):
  File "<pyshell#37>", line 1, in <module>
    next(it)
StopIteration
```

2가 나오면 StopIteration이 발생, for 반복문에 넣어서 사용가능

물론 숫자가 무작위로 생성되므로 next(it)를 호출하는 횟수도 매번 달라진다.

```
>>> import random
>>> for i in iter(lambda : random.randint(0, 5), 2):
...     print(i, end=' ')
...
3 1 4 0 5 3 3 5 0 4 1
```

iter 함수를 활용하면 if 조건문으로 매번 숫자가 2인지 검사하지 않아도 되므로 코드가 좀 더 간단해진다.

```
import random

while True:
    i = random.randint(0, 5)
    if i == 2:
        break
    print(i, end=' ')
```

39.4.2 next

1. next는 기본값을 지정할 수 있다.
2. 기본값을 지정하면 반복이 끝나더라도 StopIteration이 발생하지 않고 기본값을 출력한다.
3. **next(반복가능한객체, 기본값)**

```
>>> it = iter(range(3))
>>> next(it, 10)
0
>>> next(it, 10)
1
>>> next(it, 10)
2
>>> next(it, 10)
10
>>> next(it, 10)
10
```

0, 1, 2까지 나온 뒤에도 next(it, 10)을 호출하면 예외가 발생하지 않고 계속 10이 나온다.