

교수의 파이썬

03_1 반복가능 자료형과 반복자

창원대학교 정보통신공학과 교수 박동규

넌넌한 교수의 파이썬

03_1 반복가능 자료형과 반복자

창원대학교 정보통신공학과 교수 박동규

넌넌한 교수의 고급 파이썬

03_1 반복가능 자료형과 반복자

창원대학교 정보통신공학과 교수 박동규

넌넌한 교수의 고급 파이썬

03_1 반복가능 자료형과 반복자

창원대학교 정보통신공학과 교수 박동규

반복자란?

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

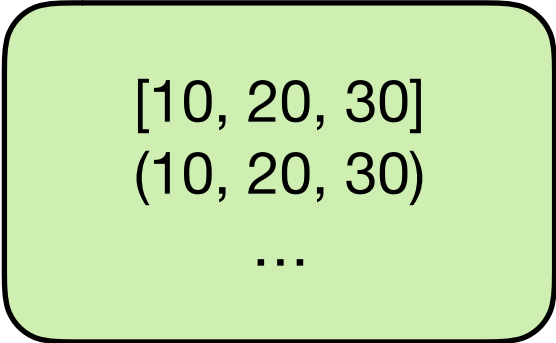
파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.



[10, 20, 30]
(10, 20, 30)
...

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

[10, 20, 30]
(10, 20, 30)
...

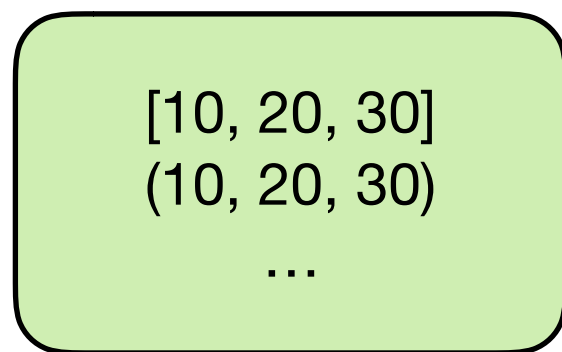
반복가능 객체 :
리스트, 튜플, range형

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서 데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나 이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고 차례차례로 다음 항목의 요소를 반환할 수 있다.



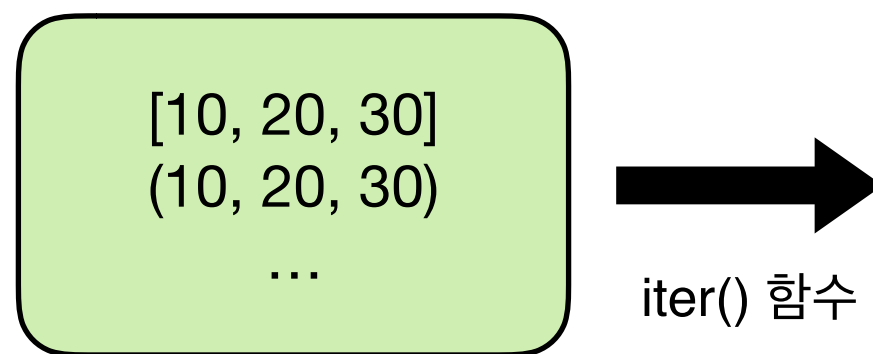
반복가능 객체 :
리스트, 튜플, range형

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.



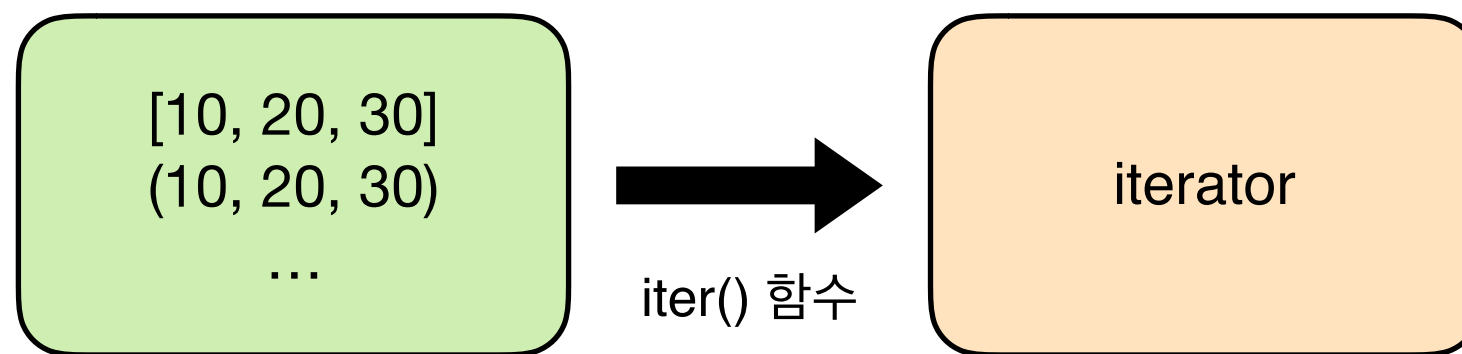
반복가능 객체 :
리스트, 튜플, range형

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.



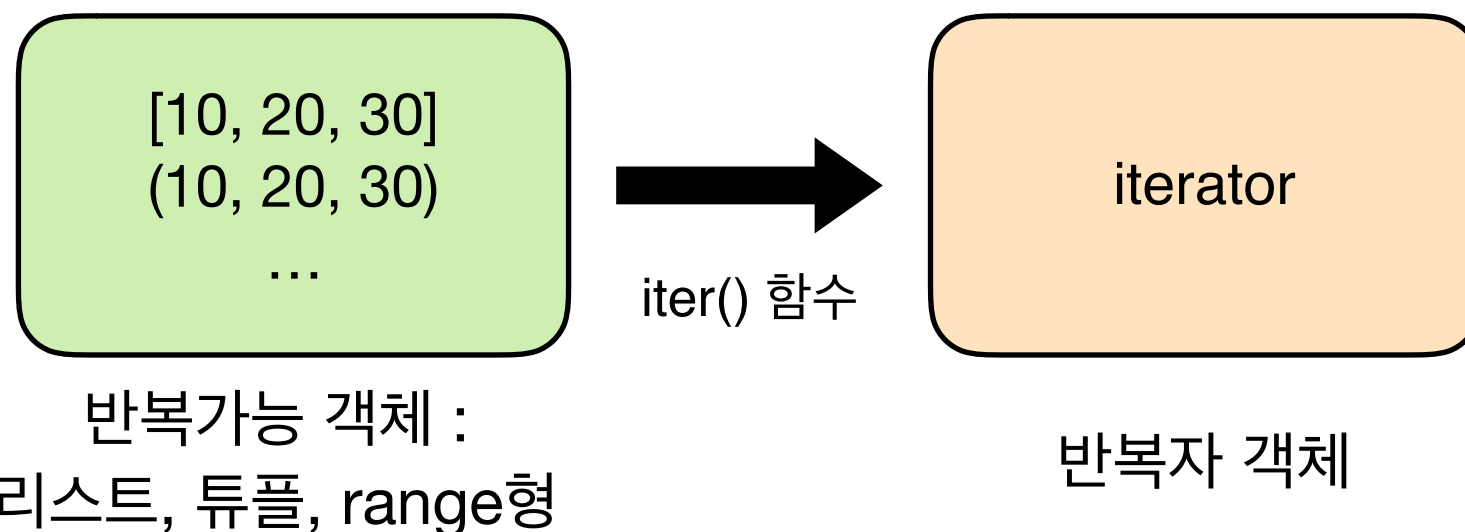
반복가능 객체 :
리스트, 튜플, range형

반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

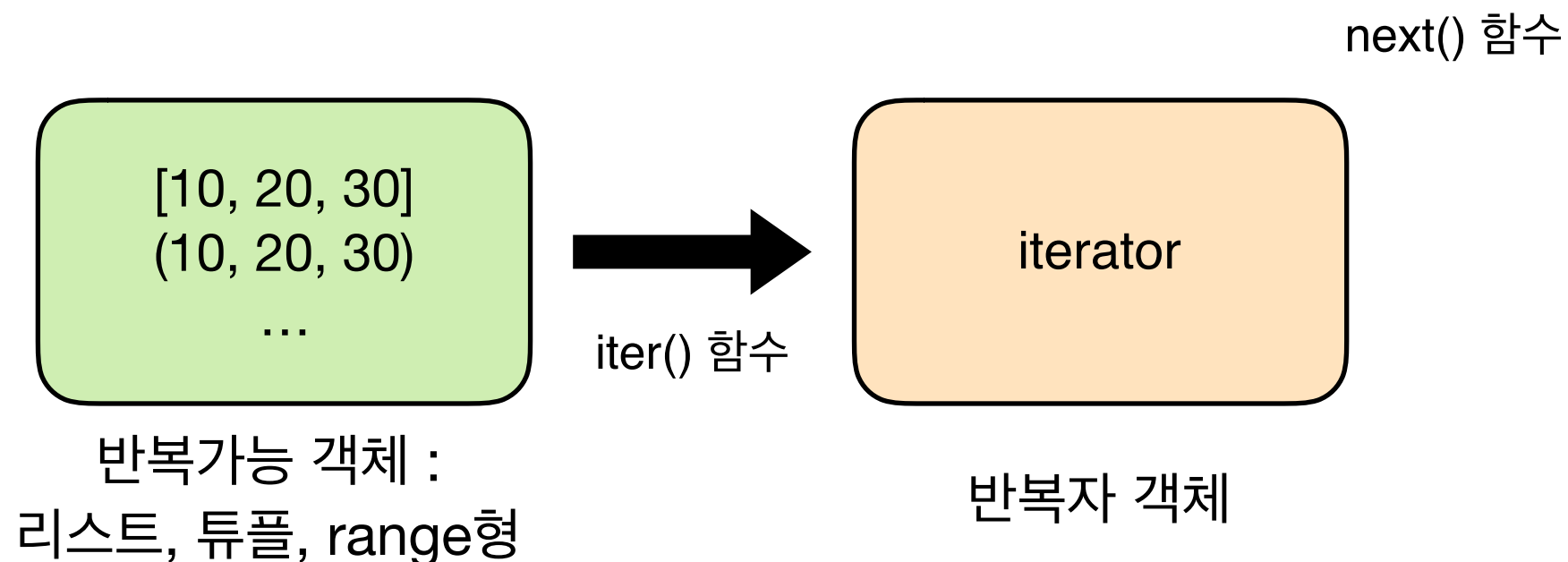


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서 데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나 이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고 차례차례로 다음 항목의 요소를 반환할 수 있다.

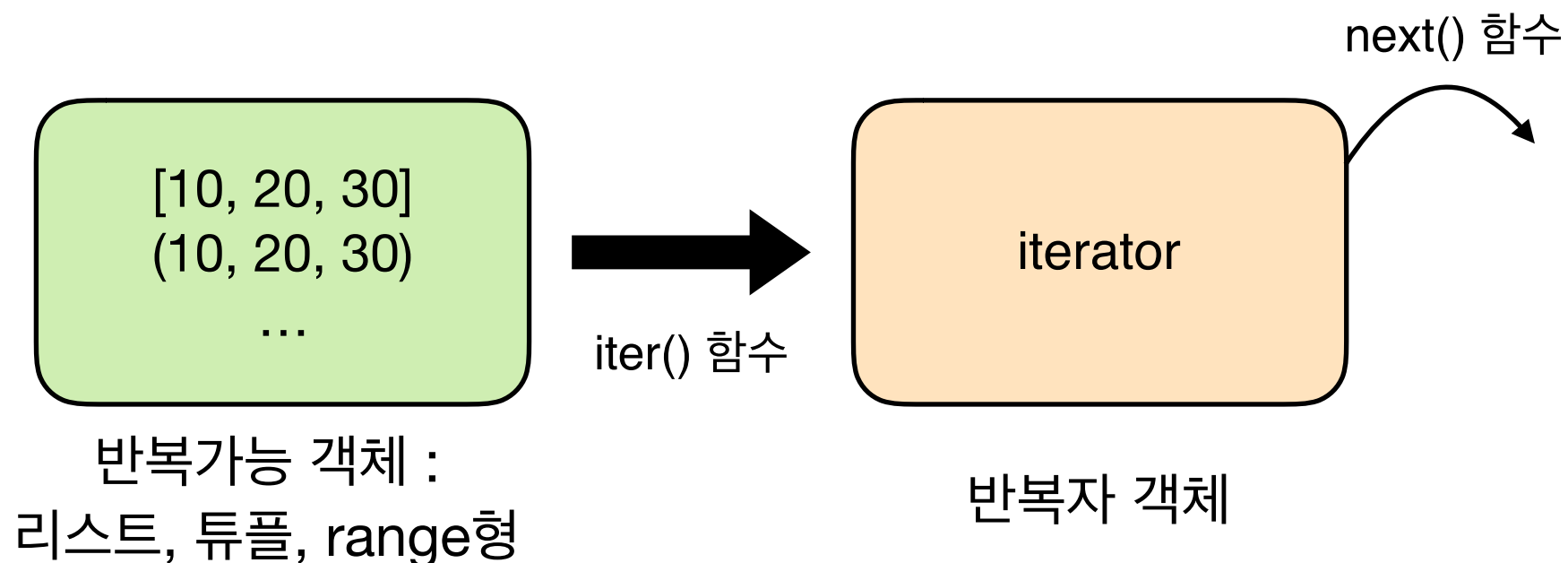


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

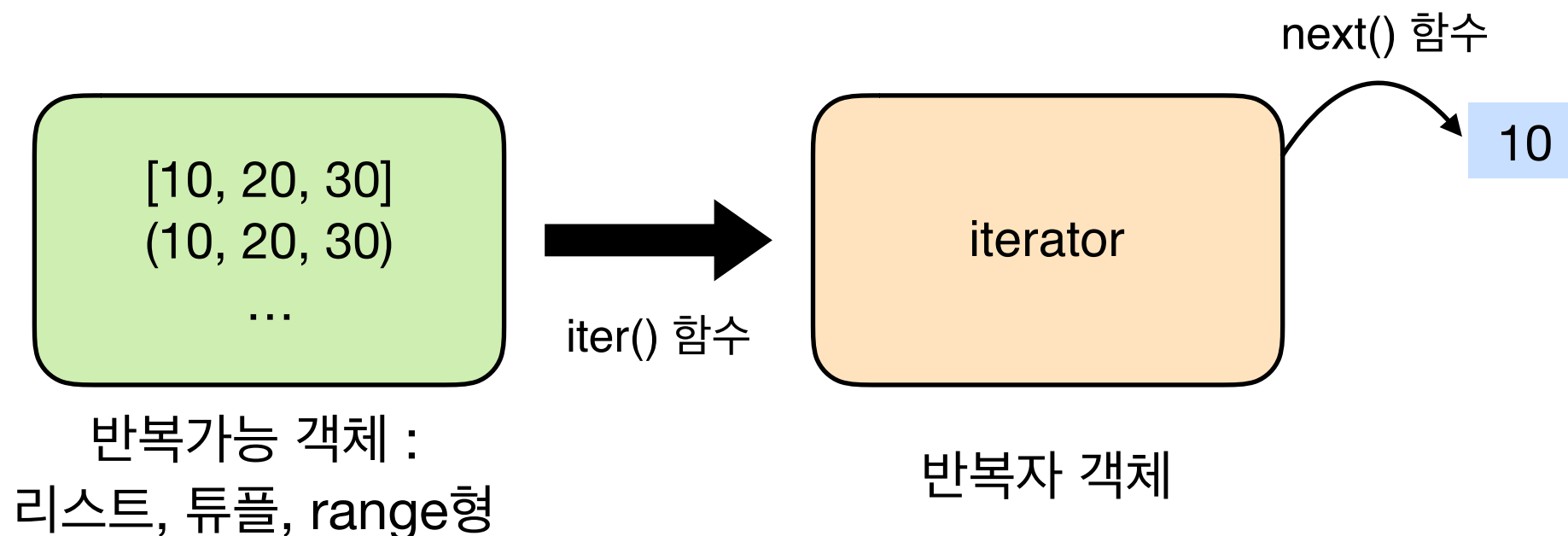


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

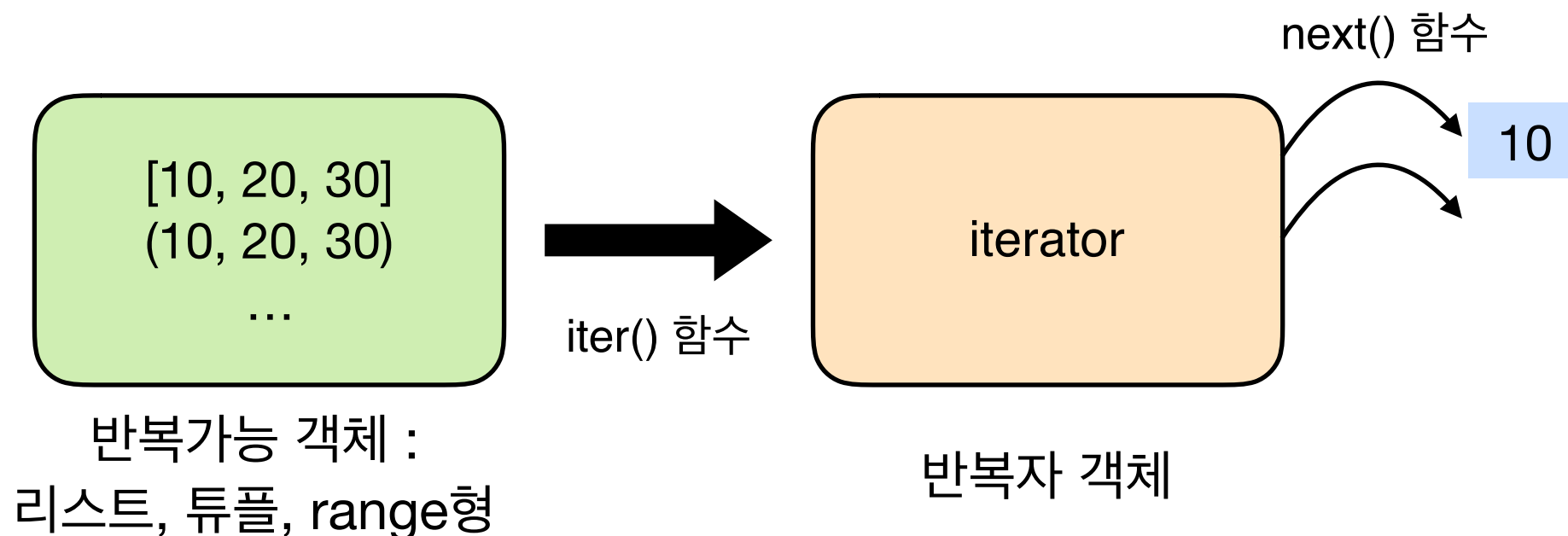


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

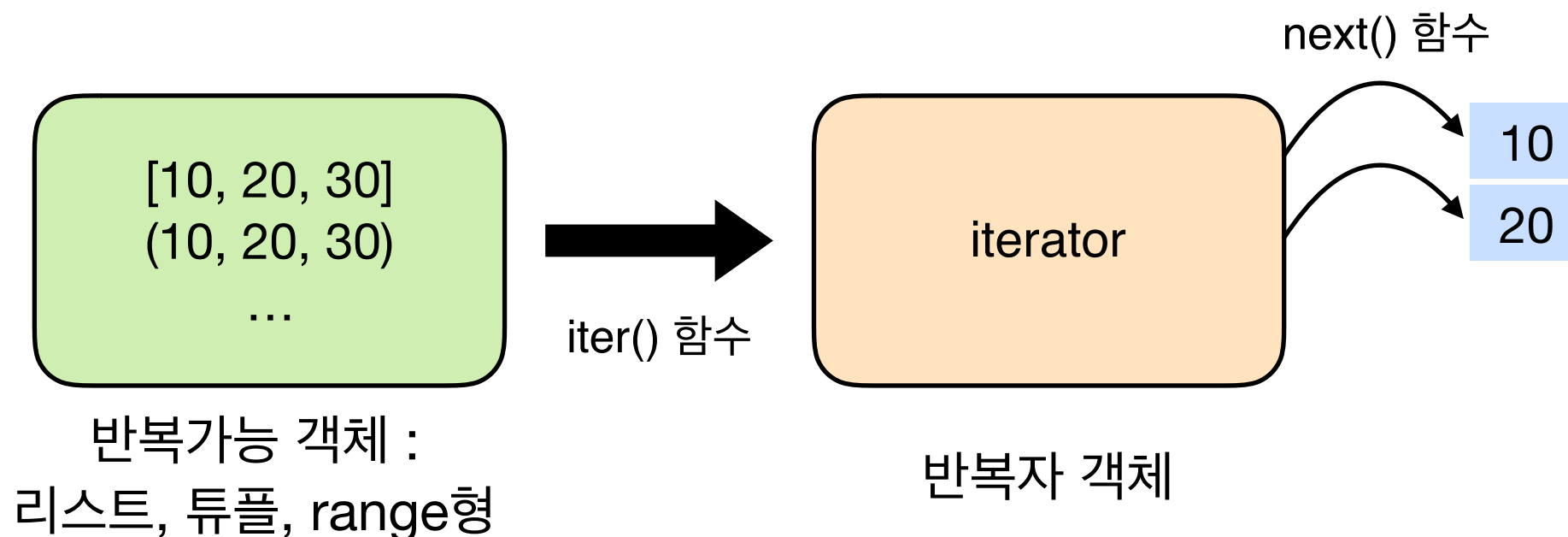


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

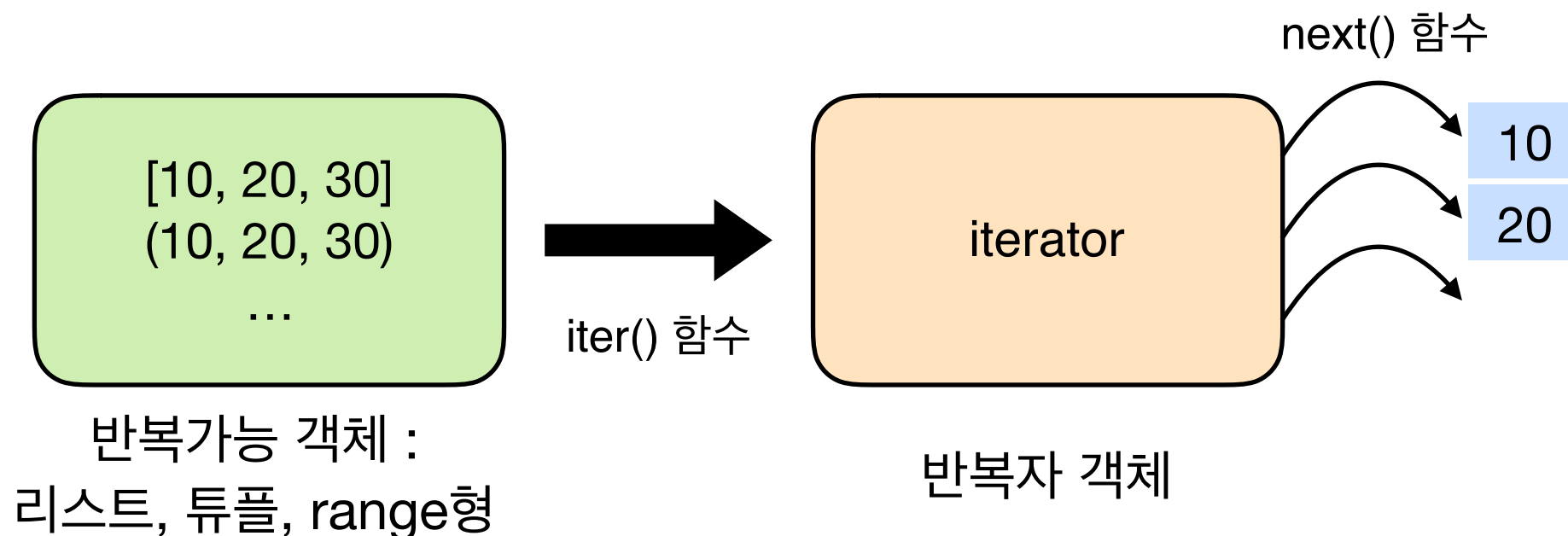


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

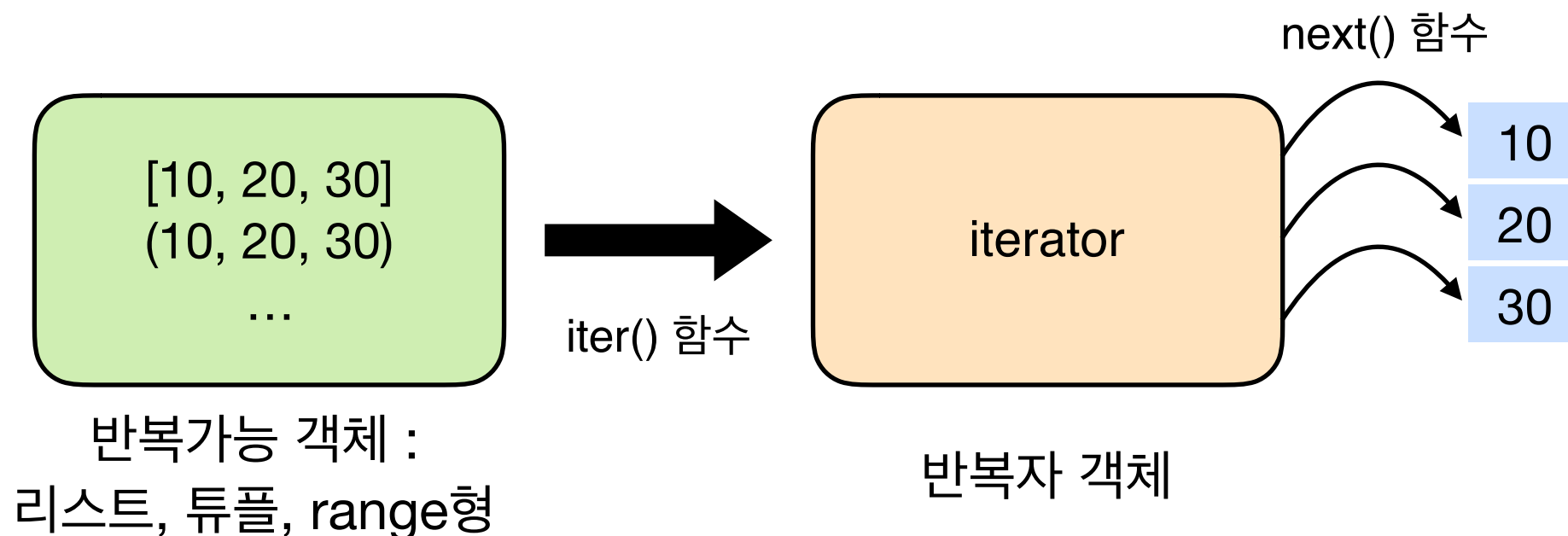


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

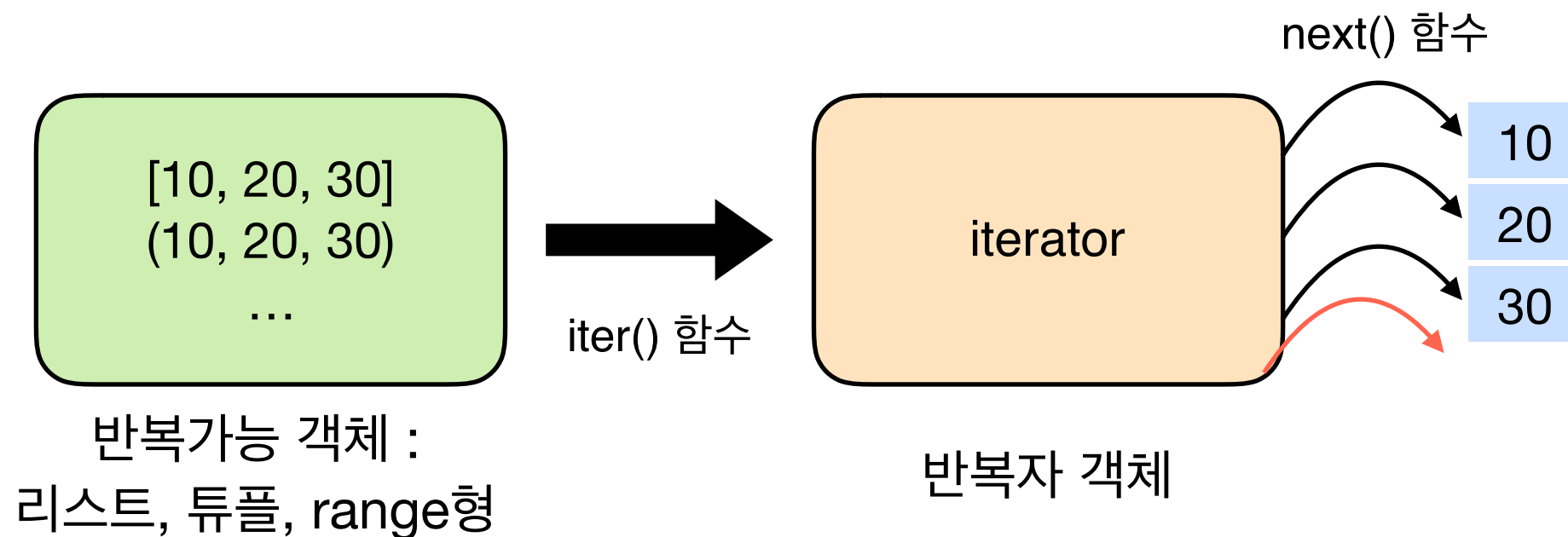


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

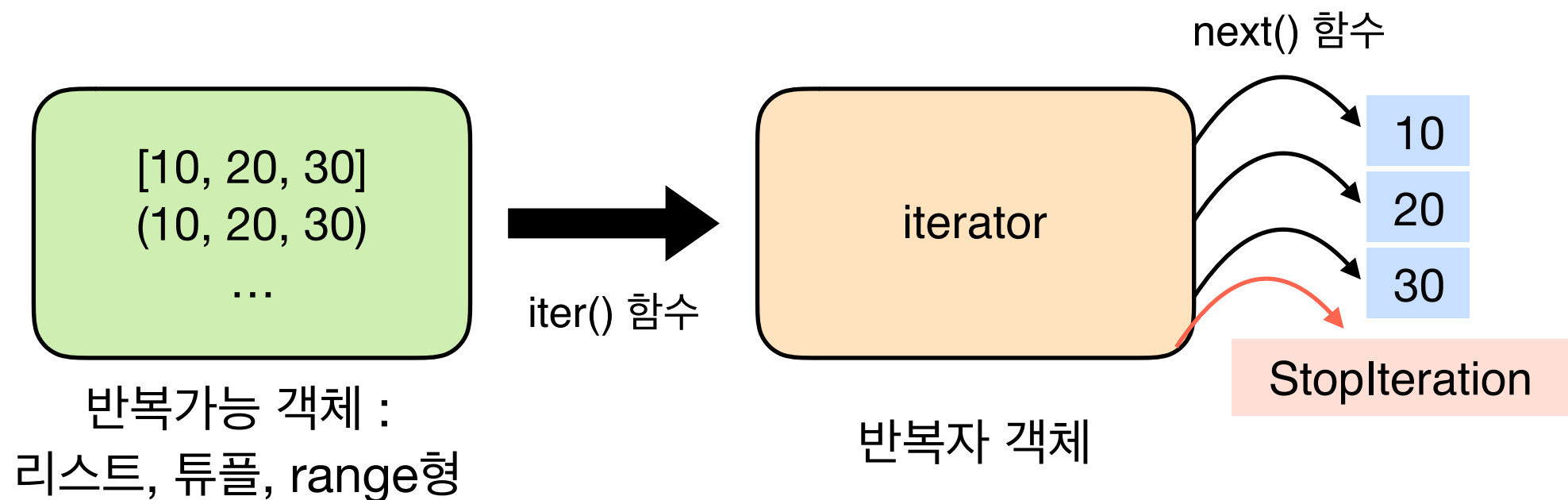


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.

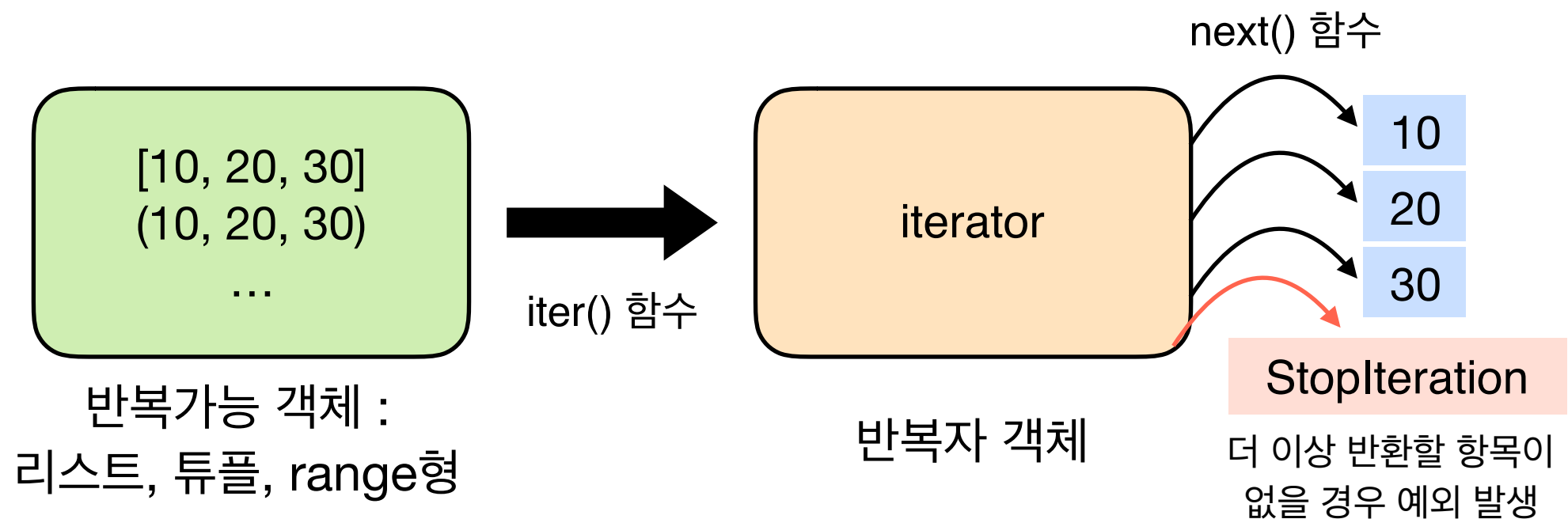


반복자란?

하나 이상의 항목이 포함되어 있는 자료구조에서
데이터를 순차적으로 꺼내어 이용할 수 있는 객체

데이터를 순차적으로 꺼낼 때에는 `next()` 라는 함수나
이 객체의 `__next__()` 라는 특수 메소드를 사용해야 한다.

파이썬의 반복자는 이렇게 `__next__()` 라는 메소드를 가지고
차례차례로 다음 항목의 요소를 반환할 수 있다.



반복가능iterable 자료형

반복가능iterable 자료형

- 반복자 자료형과 유사한 자료형
- 반복가능 자료형으로는 리스트, 딕셔너리, 튜플, 문자열, 집합, 파일, range가 있다.
- 반복가능 자료형은 파이썬 내장함수인 iter() 함수를 이용해서 반복자 객체로 만들 수 있다.

반복자 객체로 변환 가능한 리스트형과 불가능한 정수형

```
[>>> l = [10, 20, 30]
>>> l_iter = iter(l)
>>> l_iter
<list_iterator object at 0x10b74ce50>
>>> n = 100
>>> n_iter = iter(n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

반복자 객체로 변환 가능한 리스트형과 불가능한 정수형

```
[>>> l = [10, 20, 30] # 반복가능 자료형인 리스트
[>>> l_iter = iter(l)
[>>> l_iter
<list_iterator object at 0x10b74ce50>
[>>> n = 100
[>>> n_iter = iter(n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

반복자 객체로 변환 가능한 리스트형과 불가능한 정수형

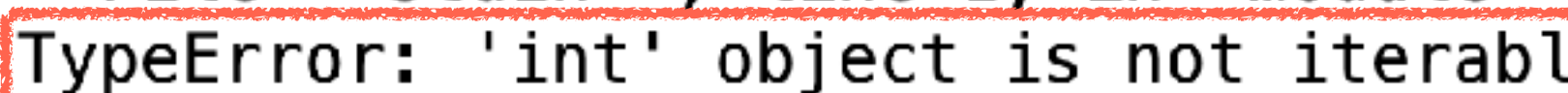
```
[>>> l = [10, 20, 30] # 반복가능 자료형인 리스트
[>>> l_iter = iter(l)
[>>> l_iter
<list_iterator object at 0x10b74ce50>
[>>> n = 100 # 반복불가능 자료형인 int
[>>> n_iter = iter(n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

반복자 객체로 변환 가능한 리스트형과 불가능한 정수형

```
[>>> l = [10, 20, 30] # 반복가능 자료형인 리스트
[>>> l_iter = iter(l)
[>>> l_iter
<list_iterator object at 0x10b74ce50>
[>>> n = 100 # 반복불가능 자료형인 int
[>>> n_iter = iter(n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

반복자 객체로 변환 가능한 리스트형과 불가능한 정수형

```
[>>> l = [10, 20, 30] # 반복가능 자료형인 리스트
[>>> l_iter = iter(l)
[>>> l_iter
<list_iterator object at 0x10b74ce50>
[>>> n = 100 # 반복불가능 자료형인 int
[>>> n_iter = iter(n)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```



반복자 객체로 변환 가능한 리스트형과 불가능한 정수형

```
[>>> l = [10, 20, 30] # 반복가능 자료형인 리스트
[>>> l_iter = iter(l)
[>>> l_iter
<list_iterator object at 0x10b74ce50>
[>>> n = 100 # 반복불가능 자료형인 int
[>>> n_iter = iter(n)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

정수 자료형 int는
반복가능 자료형이
아니기때문에
반복자 객체로 변환 불가능


```
1 # 리스트가 반복 가능 객체인가 검사
2 try:
3     l = [10, 20, 30]
4     iterator = iter(l)
5 except TypeError:
6     print('list는 iterable 객체가 아닙니다.')
7 else:
8     print('list는 iterable 객체입니다.')
9
10 # 튜플이 반복 가능 객체인가 검사
11 try:
12     t = ('홍길동', 22, 79.7)
13     iterator = iter(t)
14 except TypeError:
15     print('tuple은 iterable 객체가 아닙니다.')
16 else:
17     print('tuple은 iterable 객체입니다.')
18
19 # 정수형이 반복 가능한 객체인가 검사
20 try:
21     n = 100
22     iterator = iter(n)
23 except TypeError:
24     print('n은 iterable 객체가 아닙니다.')
25 else:
26     print('n은 iterable 객체입니다.')
```

```
1 # 리스트가 반복 가능 객체인가 검사
2 try:
3     l = [10, 20, 30]
4     iterator = iter(l)
5 except TypeError:
6     print('list는 iterable 객체가 아닙니다.')
7 else:
8     print('list는 iterable 객체입니다.')
9
10 # 튜플이 반복 가능 객체인가 검사
11 try:
12     t = ('홍길동', 22, 79.7)
13     iterator = iter(t)
14 except TypeError:
15     print('tuple은 iterable 객체가 아닙니다.')
16 else:
17     print('tuple은 iterable 객체입니다.')
18
19 # 정수형이 반복 가능한 객체인가 검사
20 try:
21     n = 100
22     iterator = iter(n)
23 except TypeError:
24     print('n은 iterable 객체가 아닙니다.')
25 else:
26     print('n은 iterable 객체입니다.')
```

```
1 # 리스트가 반복 가능 객체인가 검사
2 try:
3     l = [10, 20, 30]
4     iterator = iter(l)
5 except TypeError:
6     print('list는 iterable 객체가 아닙니다.')
7 else:
8     print('list는 iterable 객체입니다.')
9
10 # 튜플이 반복 가능 객체인가 검사
11 try:
12     t = ('홍길동', 22, 79.7)
13     iterator = iter(t)
14 except TypeError:
15     print('tuple은 iterable 객체가 아닙니다.')
16 else:
17     print('tuple은 iterable 객체입니다.')
18
19 # 정수형이 반복 가능한 객체인가 검사
20 try:
21     n = 100
22     iterator = iter(n)
23 except TypeError:
24     print('n은 iterable 객체가 아닙니다.')
25 else:
26     print('n은 iterable 객체입니다.')
```

```
1 # 리스트가 반복 가능 객체인가 검사
2 try:
3     l = [10, 20, 30]
4     iterator = iter(l)
5 except TypeError:
6     print('list는 iterable 객체가 아닙니다.')
7 else:
8     print('list는 iterable 객체입니다.')
9
10 # 튜플이 반복 가능 객체인가 검사
11 try:
12     t = ('홍길동', 22, 79.7)
13     iterator = iter(t)
14 except TypeError:
15     print('tuple은 iterable 객체가 아닙니다.')
16 else:
17     print('tuple은 iterable 객체입니다.')
18
19 # 정수형이 반복 가능한 객체인가 검사
20 try:
21     n = 100
22     iterator = iter(n)
23 except TypeError:
24     print('n은 iterable 객체가 아닙니다.')
25 else:
26     print('n은 iterable 객체입니다.')
```

```

1 # 리스트가 반복 가능 객체인가 검사
2 try:
3     l = [10, 20, 30]
4     iterator = iter(l)
5 except TypeError:
6     print('list는 iterable 객체가 아닙니다.')
7 else:
8     print('list는 iterable 객체입니다.')
9
10 # 튜플이 반복 가능 객체인가 검사
11 try:
12     t = ('홍길동', 22, 79.7)
13     iterator = iter(t)
14 except TypeError:
15     print('tuple은 iterable 객체가 아닙니다.')
16 else:
17     print('tuple은 iterable 객체입니다.')
18
19 # 정수형이 반복 가능한 객체인가 검사
20 try:
21     n = 100
22     iterator = iter(n)
23 except TypeError:
24     print('n은 iterable 객체가 아닙니다.')
25 else:
26     print('n은 iterable 객체입니다.')

```

실행결과

list는 iterable 객체입니다.
 tuple은 iterable 객체입니다.
 n은 iterable 객체가 아닙니다.

```

1 # 리스트가 반복 가능 객체인가 검사
2 try:
3     l = [10, 20, 30]
4     iterator = iter(l)
5 except TypeError:
6     print('list는 iterable 객체가 아닙니다.')
7 else:
8     print('list는 iterable 객체입니다.')
9
10 # 튜플이 반복 가능 객체인가 검사
11 try:
12     t = ('홍길동', 22, 79.7)
13     iterator = iter(t)
14 except TypeError:
15     print('tuple은 iterable 객체가 아닙니다.')
16 else:
17     print('tuple은 iterable 객체입니다.')
18
19 # 정수형이 반복 가능한 객체인가 검사
20 try:
21     n = 100
22     iterator = iter(n)
23 except TypeError:
24     print('n은 iterable 객체가 아닙니다.')
25 else:
26     print('n은 iterable 객체입니다.')

```

실행결과

list는 iterable 객체입니다.
tuple은 iterable 객체입니다.
n은 iterable 객체가 아닙니다.

Lab

next() 함수를 사용한 반복자 객체의 요소 추출

```
[>>> lst= [10, 20, 30]
[>>> l_iter = iter(lst)
[>>> type(l_iter)
<class 'list_iterator'>
[>>> next(l_iter)
10
[>>> next(l_iter)
20
[>>> next(l_iter)
30
[>>> next(l_iter)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```


next() 함수를 사용한 반복자 객체의 요소 추출


```
[>>> lst= [10, 20, 30]
[>>> l_iter = iter(lst)      # 리스트 형 객체를 반복자로 만듦
[>>> type(l_iter)
<class 'list_iterator'>
[>>> next(l_iter)
10
[>>> next(l_iter)
20
[>>> next(l_iter)
30
[>>> next(l_iter)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

next() 함수를 사용한 반복자 객체의 요소 추출

```
[>>> lst= [10, 20, 30]
[>>> l_iter = iter(lst)      # 리스트 형 객체를 반복자로 만듦
[>>> type(l_iter)
<class 'list_iterator'>
[>>> next(l_iter)
10
[>>> next(l_iter)
20
[>>> next(l_iter)
30
[>>> next(l_iter)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```


next() 함수를 사용한 반복자 객체의 요소 추출

```
[>>> lst= [10, 20, 30]
[>>> l_iter = iter(lst)      # 리스트 형 객체를 반복자로 만듦
[>>> type(l_iter)
<class 'list_iterator'>
[>>> next(l_iter)
10
[>>> next(l_iter)
20
[>>> next(l_iter)
30
[>>> next(l_iter)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```



next() 함수를 사용한 반복자 객체의 요소 추출

```
[>>> lst= [10, 20, 30]
[>>> l_iter = iter(lst)      # 리스트 형 객체를 반복자로 만듦
[>>> type(l_iter)
<class 'list_iterator'>
[>>> next(l_iter)
10
[>>> next(l_iter)
20
[>>> next(l_iter)
30
[>>> next(l_iter)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```



모든 요소를 반환한 반복자 객체에
next() 함수를 호출하면
StopIteration 예외 생성

__next__() 메소드를 사용한 반복자 객체의 요소 추출

```
>>> lst = [10, 20, 30]
>>> l_iter = iter(lst)
>>> l_iter.__next__()
10
>>> l_iter.__next__()
20
>>> l_iter.__next__()
30
>>> l_iter.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

__next__() 메소드를 사용한 반복자 객체의 요소 추출


```
>>> lst = [10, 20, 30]
>>> l_iter = iter(lst) # 리스트 형 객체를 반복자로 만듦
>>> l_iter.__next__()
10
>>> l_iter.__next__()
20
>>> l_iter.__next__()
30
>>> l_iter.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

__next__() 메소드를 사용한 반복자 객체의 요소 추출

```
>>> lst = [10, 20, 30]
>>> l_iter = iter(lst) # 리스트 형 객체를 반복자로 만듦
>>> l_iter.__next__()
10
>>> l_iter.__next__()
20
>>> l_iter.__next__()
30
>>> l_iter.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

__next__() 메소드를 사용한 반복자 객체의 요소 추출

```
>>> lst = [10, 20, 30]
>>> l_iter = iter(lst) # 리스트 형 객체를 반복자로 만듦
>>> l_iter.__next__()
10
>>> l_iter.__next__()
20
>>> l_iter.__next__()
30
>>> l_iter.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```



__next__() 메소드를 사용한 반복자 객체의 요소 추출

```
>>> lst = [10, 20, 30]
>>> l_iter = iter(lst) # 리스트 형 객체를 반복자로 만듦
>>> l_iter.__next__()
10
>>> l_iter.__next__()
20
>>> l_iter.__next__()
30
>>> l_iter.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

‘객체명.__next__()’를 써주면
next() 함수의 호출과
동일하게 작용

__next__() 메소드를 사용한 반복자 객체의 요소 추출

```
>>> lst = [10, 20, 30]
>>> l_iter = iter(lst) # 리스트 형 객체를 반복자로 만듦
>>> l_iter.__next__()
10
>>> l_iter.__next__()
20
>>> l_iter.__next__()
30
>>> l_iter.__next__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

‘객체명.__next__()’를 써주면
next() 함수의 호출과
동일하게 작용

range 형 객체를 반복자 객체로 만들기

range 형 객체를 반복자 객체로 만들기

range() 함수는 range 형 객체를 만들어준다.
이 range 형 객체는 iter() 함수를 통해
range_iterator 형으로 변환시킬 수 있는 **반복가능 객체**이다.

range 형 객체를 반복자 객체로 만들기

range() 함수는 range 형 객체를 만들어준다.
이 range 형 객체는 iter() 함수를 통해
range_iterator 형으로 변환시킬 수 있는 **반복가능 객체**이다.

range_iterator 형 객체는 next() 함수를 통해 다음 항목으로 접근할 수 있다

range 형 객체를 반복자 객체로 만들기

range() 함수는 range 형 객체를 만들어준다.
이 range 형 객체는 iter() 함수를 통해
range_iterator 형으로 변환시킬 수 있는 **반복가능 객체**이다.

range_iterator 형 객체는 next() 함수를 통해 다음 항목으로 접근할 수 있다

```
[>>> type(range(3))
<class 'range'>
[>>> r_iter = iter(range(3))
[>>> type(r_iter)
<class 'range_iterator'>
[>>> next(r_iter)
0
[>>> next(r_iter)
1
```

range 형 객체를 반복자 객체로 만들기

range() 함수는 range 형 객체를 만들어준다.
이 range 형 객체는 iter() 함수를 통해
range_iterator 형으로 변환시킬 수 있는 **반복가능 객체**이다.

range_iterator 형 객체는 next() 함수를 통해 다음 항목으로 접근할 수 있다

```
[>>> type(range(3))
<class 'range'>
[>>> r_iter = iter(range(3)) # range 형 객체를 반복자로 만듦
[>>> type(r_iter)
<class 'range_iterator'>
[>>> next(r_iter)
0
[>>> next(r_iter)
1
```

range 형 객체를 반복자 객체로 만들기

range() 함수는 range 형 객체를 만들어준다.
이 range 형 객체는 iter() 함수를 통해
range_iterator 형으로 변환시킬 수 있는 **반복가능 객체**이다.

range_iterator 형 객체는 next() 함수를 통해 다음 항목으로 접근할 수 있다

```
[>>> type(range(3))  
<class 'range'>  
[>>> r_iter = iter(range(3)) # range 형 객체를 반복자로 만들  
[>>> type(r_iter)  
<class 'range_iterator'>  
[>>> next(r_iter) # 반복자이므로 next( ) 함수를 사용할 수 있음  
0  
[>>> next(r_iter)  
1
```


iterator 객체와
for - in range()구문

iterator 객체와 for - in range()구문

range 형 객체의 for - in 구문에서의 사용법

iterator 객체와 for - in range()구문

range 형 객체의 for - in 구문에서의 사용법

```
>>> for i in range(5) :  
...     print(i, end = ' ' )  
...  
0 1 2 3 4
```

iterator 객체와 for - in range()구문

range 형 객체의 for - in 구문에서의 사용법

```
>>> for i in range(5) :  
...     print(i, end = ' ' )  
...  
0 1 2 3 4
```

위의 대화창 실습 코드와 같이 iterator 객체는
for - in 구문을 통해서 반복적으로
하나씩 데이터를 꺼내서 처리할 수 있다.

Lab

정리

- 파이썬은 반복가능 자료형(iterable)과 반복불가능 자료형이 있음
- 반복가능 자료형으로는 list, dict, set, str, tuple, bytes, range가 있다.
- 반복가능 객체를 iter() 함수를 사용하여 반복자 객체(iterator)로 만들 수 있다
- next() 내장 함수나 __next__() 메소드를 이용하여 반복자 객체내의 원소를 하나하나 뽑아낼 수 있다

감사합니다.