윤소영
imbgirl@naver.com

Lambda & built-in functions

Lambda, zip, map, filter, enumerate

lambda 함수는 언제 사용할까?

- 이름 없는 function을 '한 줄'로 작성할 수 있는 'expression' 작성 방법이다

```
def add1(a, b):
    return a + b
x = add1(10, 20)
```

```
add2 = lambda a, b : a + b
y = add2(10, 20)
```

lambda

```
y = (lambda a, b : a + b)(10, 20)
```

한 줄의 간단한 함수가 필요한 경우

프로그램의 가독성을 높이기 위해

주로 다른 함수의 argument로
간단한 동작을 하는 함수 전달 시 사용

lambda 함수의 특징

```
def function_name ( parameter_list ) : suite
```

```
lambda parameter_list : expression
```

def 함수	lambda 함수

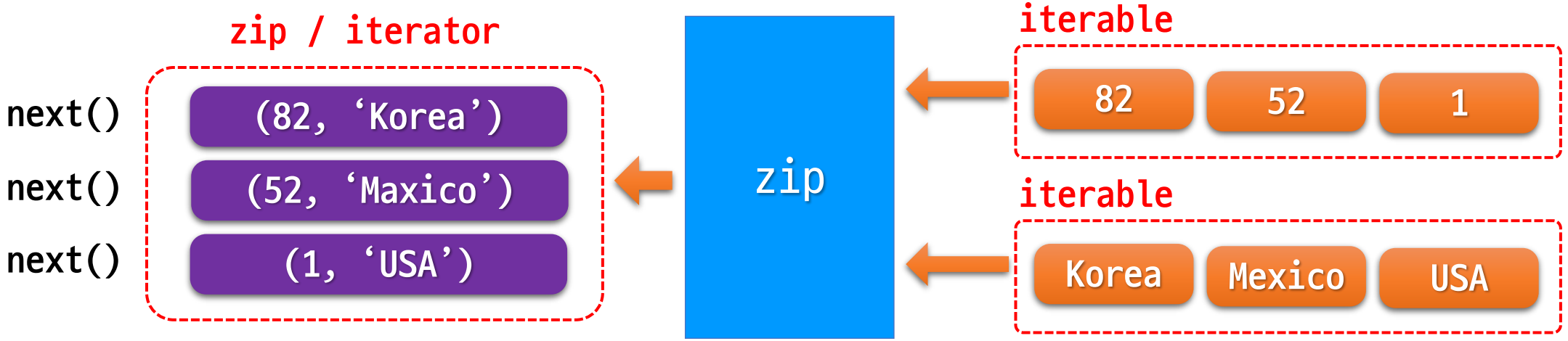
`def add(a, b):`
`  return a + b`
`x = add(10, 20)`
	`y = (lambda a, b : a + b)(10, 20)`

statement	expression
statement, expression을 모두 사용	expression만 사용
return문을 사용하여 결과 반환	expression의 실행 결과가 반환 됨 여러 개 반환 시 Container로 반환
namespace에 name이 등록됨	namespace에 name이 등록되지 않음

zip 함수

`zip(*iterable)` `-> zip / iterator`

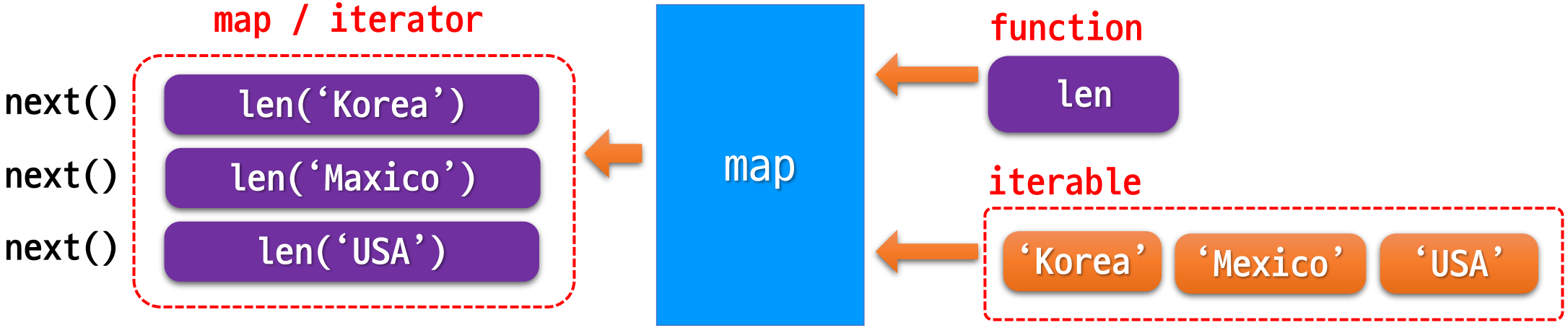
- 동일 item 개수로 이루어진 iterable을 대상으로 함
- next()할 때마다, 각 iterable의 동일 위치 item들을 모아 tuple로 반환
- iterable의 item 개수 만큼 next()를 사용할 수 있음



map 함수

`map(function, *iterable)` `-> map / iterator`

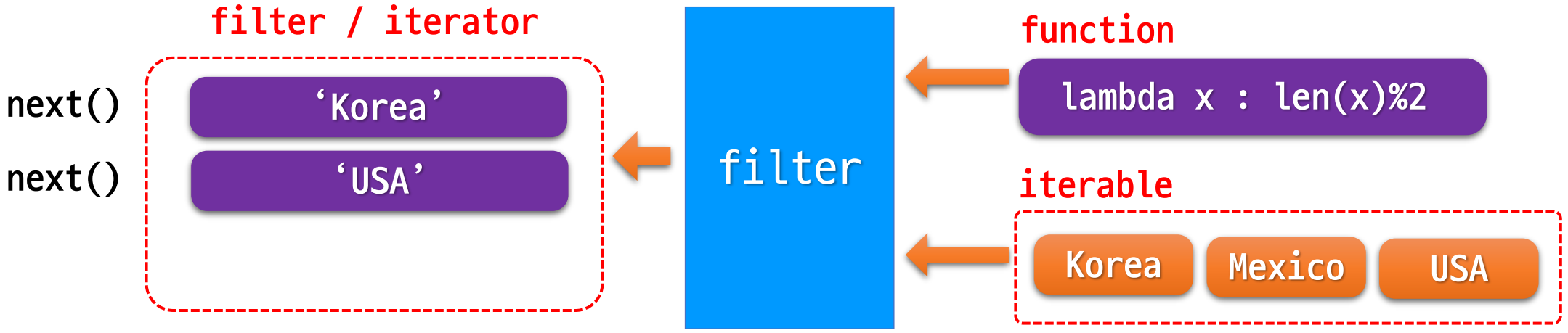
- iterable의 각 item에 function이 적용되어 map 객체 반환
- function의 parameter 개수 == iterable의 개수 (NOT item의 개수)
- next()할 때마다, function의 동작 결과가 반환됨
- iterable의 item 개수 만큼 next()를 사용할 수 있음



filter 함수

`filter(function, iterable)` `-> filter / iterator`

- iterable의 각 item에 function이 적용되어 filter 객체 반환
- next()할 때마다, function의 결과가 True인 item을 반환
- iterable의 item 개수 보다 next() 사용 횟수가 적거나 같을 수 있음



enumerate 함수

`enumerate(iterable, start=0)` `-> enumerate / iterator`

- iterable의 item에 순서 번호가 함께 필요할 때 사용함
- next()할 때마다, (index, item)의 tuple을 반환
- iterable의 item 개수 만큼 next()를 사용할 수 있음