

파이썬 프로그래밍 강의노트 #16

람다(lamda) 표현식 활용

람다(lamda)

람다 표현식 사용하기

- 람다 표현식은 식 형태로 되어 있다고 해서 람다 표현식(lambda expression)이라고 부름
- 람다 표현식은 함수를 간편하게 작성할 수 있어서 다른 함수의 인수로 넣을 때 주로 사용함

람다(lambda)

람다 표현식으로 함수 만들기

```
>>> def plus_ten(x):  
...     return x + 10  
...  
>>> plus_ten(1)  
11
```

- plus_ten 함수를 람다 표현식을 작성해보자
- 람다 표현식은 다음과 같이 lambda에 매개변수를 지정하고 :(콜론) 뒤에 반환값으로 사용할 식을 지정함
- lambda 매개변수들: 식

```
>>> lambda x: x + 10  
<function <lambda> at 0x02C27270>
```

람다(lamda)

람다 표현식으로 함수 만들기

- 실행을 해보면 함수 객체가 나오는데, 이 상태로는 함수를 호출할 수 없음
- 람다 표현식은 이름이 없는 함수를 만들기 때문임
- 람다 표현식을 익명 함수(anonymous function)로 부르기도 함
- lambda로 만든 익명 함수를 호출하려면 다음과 같이 람다 표현식을 변수에

```
>>> plus_ten = lambda x: x + 10
>>> plus_ten(1)
11
```

- 람다 표현식을 살펴보면 lambda x: x + 10은 매개변수 x 하나를 받고, x에 10을 더해서 반환한다는 뜻임
- 매개변수, 연산자, 값 등을 조합한 식으로 반환값을 만드는 방식임

람다(lamda)

▼ 그림 32-1 def로 만든 함수와 람다 표현식



람다(lamda)

람다 표현식 자체를 호출하기

- 람다 표현식은 변수에 할당하지 않고 람다 표현식 자체를 바로 호출할 수 있음
 - (lambda 매개변수들: 식)(인수들)

```
>>> (lambda x: x + 10)(1)
11
```

람다(lamda)

람다 표현식 안에서는 변수를 만들 수 없다

- 반환값 부분은 변수 없이 식 한 줄로 표현할 수 있어야 함
- 변수가 필요한 코드일 경우에는 def로 함수를 작성하는 것이 좋음

```
>>> (lambda x: y = 10; x + y)(1)
SyntaxError: invalid syntax
```

- 람다 표현식 바깥에 있는 변수는 사용할 수 있음
- 다음은 매개변수 x와 람다 표현식 바깥에 있는 변수 y를 더해서 반환함

```
>>> y = 10
>>> (lambda x: x + y)(1)
11
```

람다(lamda)

람다 표현식을 인수로 사용하기

- 람다 표현식을 사용하기 전에 먼저 def로 함수를 만들어서 map을 사용해보자

```
>>> def plus_ten(x):  
...     return x + 10  
...  
>>> list(map(plus_ten, [1, 2, 3]))  
[11, 12, 13]
```

- plus_ten처럼 함수를 직접 만들어서 넣어도 됨
- 이제 람다 표현식으로 함수를 만들어서 map에 넣어보자

```
>>> list(map(lambda x: x + 10, [1, 2, 3]))  
[11, 12, 13]
```


람다(lamda) : map,filter,reduce 활용

람다 표현식에 조건부 표현식 사용하기

- `lambda` 매개변수들: 식1 `if` 조건식 `else` 식2
- 다음은 `map`을 사용하여 리스트 `a`에서 3의 배수를 문자열로 변환함

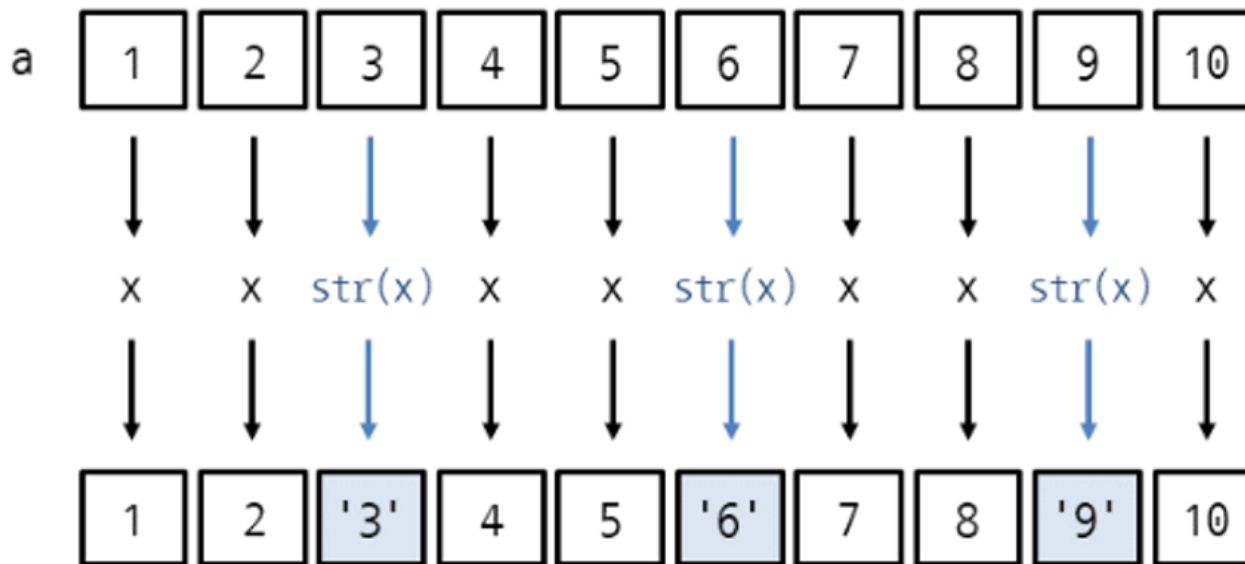
```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(map(lambda x: str(x) if x % 3 == 0 else x, a))
[1, 2, '3', 4, 5, '6', 7, 8, '9', 10]
```

- `map`은 리스트의 요소를 각각 처리하므로 `lambda`의 반환값도 요소라야 함
- 여기서는 요소가 3의 배수일 때는 `str(x)`로 요소를 문자열로 만들어서 반환했고, 3의 배수가 아닐 때는 `x`로 요소를 그대로 반환함

람다(lamda) : map,filter,reduce 활용

▼ 그림 32-2 map에 람다 표현식 사용하기

```
list(map(lambda x: str(x) if x % 3 == 0 else x, a))
```



람다(lamda) : map,filter,reduce 활용

람다 표현식에 조건부 표현식 사용하기

- 람다 표현식 안에서 조건부 표현식 if, else를 사용할 때는 :(콜론)을 붙이지 않음
- if, else와 문법이 다르므로 주의해야 함
- 조건부 표현식은 식1 if 조건식 else 식2 형식으로 사용하며 식1은 조건식이 참일 때, 식2는 조건식이 거짓일 때 사용할 식임
- 특히 람다 표현식에서 if를 사용했다면 반드시 else를 사용해야 함
- 다음과 같이 if만 사용하면 문법 에러가 발생하므로 주의해야 함

```
>>> list(map(lambda x: str(x) if x % 3 == 0, a))  
SyntaxError: invalid syntax
```

람다(lamda) : map,filter,reduce 활용

람다 표현식에 조건부 표현식 사용하기

- 람다 표현식 안에서는 elif를 사용할 수 없음
- 조건부 표현식은 식1 if 조건식1 else 식2 if 조건식2 else 식3 형식처럼 if를 연속으로 사용해야 함
- lambda 매개변수들: 식1 if 조건식1 else 식2 if 조건식2 else 식3

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(map(lambda x: str(x) if x == 1 else float(x) if x == 2 else x + 10, a))
['1', 2.0, 13, 14, 15, 16, 17, 18, 19, 20]
```

람다(lamda) : map,filter,reduce 활용

람다 표현식에 조건부 표현식 사용하기

- 별로 복잡하지 않은 조건인데도 알아보기가 힘든 경우에는 억지로 람다 표현식을 사용하기 보다는 그냥 def로 함수를 만들고 if, elif, else를 사용하는 것을 권장함

```
>>> def f(x):  
...     if x == 1:  
...         return str(x)  
...     elif x == 2:  
...         return float(x)  
...     else:  
...         return x + 10  
...  
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
>>> list(map(f, a))  
['1', 2.0, 13, 14, 15, 16, 17, 18, 19, 20]
```

람다(lamda) : map,filter,reduce 활용

map에 객체를 여러 개 넣기

- 다음은 두 리스트의 요소를 곱해서 새 리스트를 만듦

```
>>> a = [1, 2, 3, 4, 5]
>>> b = [2, 4, 6, 8, 10]
>>> list(map(lambda x, y: x * y, a, b))
[2, 8, 18, 32, 50]
```

- 람다 표현식의 매개변수 개수에 맞게 반복 가능한 객체도 콤마로 구분해서 넣어주면 됨

람다(lamda) : map,filter,reduce 활용

filter 사용하기

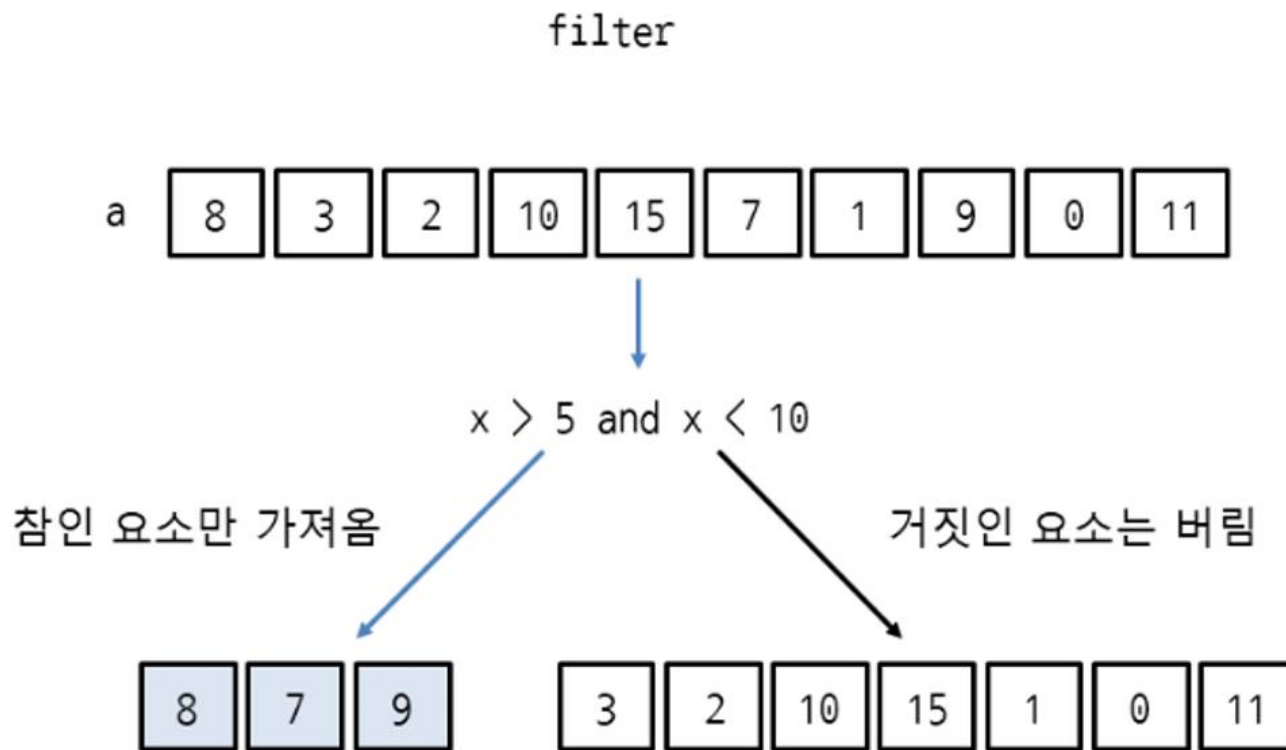
- filter는 반복 가능한 객체에서 특정 조건에 맞는 요소만 가져오는데, filter에 지정한 함수의 반환값이 True일 때만 해당 요소를 가져옴
- `filter(함수, 반복가능한객체)`

```
>>> def f(x):  
...     return x > 5 and x < 10  
...  
>>> a = [8, 3, 2, 10, 15, 7, 1, 9, 0, 11]  
>>> list(filter(f, a))  
[8, 7, 9]
```

- filter는 `x > 5 and x < 10`의 결과가 참인 요소만 가져오고 거짓인 요소는 버림

람다(lamda) : map,filter,reduce 활용

▼ 그림 32-3 filter 함수



람다(lamda) : map,filter,reduce 활용

filter 사용하기

- 그럼 함수 f를 람다 표현식으로 만들어서 filter에 넣어보자

```
>>> a = [8, 3, 2, 10, 15, 7, 1, 9, 0, 11]
>>> list(filter(lambda x: x > 5 and x < 10, a))
[8, 7, 9]
```

람다(lamda) : map,filter,reduce 활용

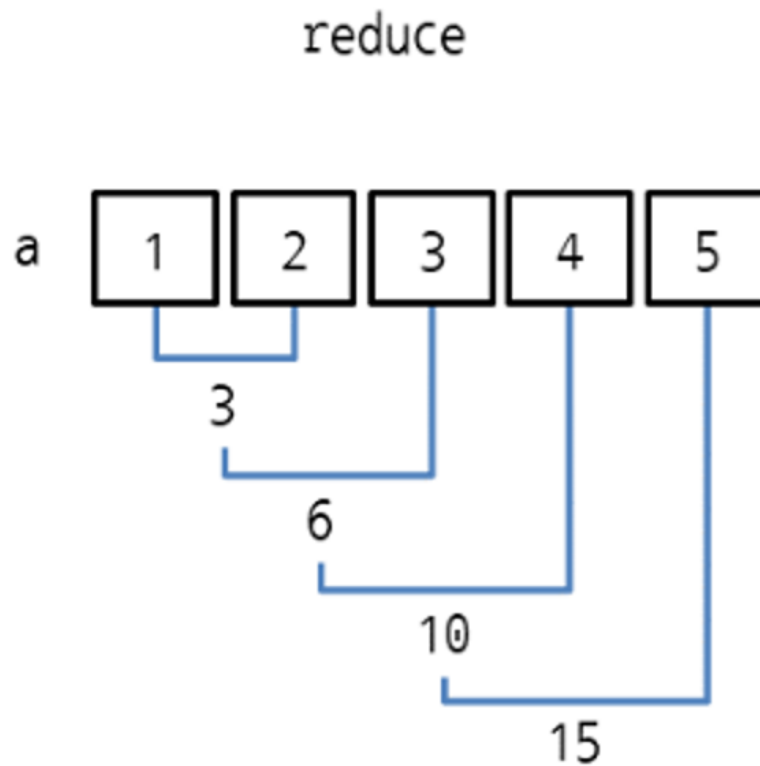
reduce 사용하기

- reduce는 반복 가능한 객체의 각 요소를 지정된 함수로 처리한 뒤 이전 결과와 누적해서 반환하는 함수임(reduce는 파이썬 3부터 내장 함수가 아님. 따라서 functools 모듈에서 reduce 함수를 가져와야 함)
 - `from functools import reduce`
 - `reduce(함수, 반복가능한객체)`
- 다음은 리스트에 저장된 요소를 순서대로 더한 뒤 누적된 결과를 반환함

```
>>> def f(x, y):  
...     return x + y  
...  
>>> a = [1, 2, 3, 4, 5]  
>>> from functools import reduce  
>>> reduce(f, a)  
15
```

람다(lamda) : map,filter,reduce 활용

▼ 그림 32-4 reduce 함수



람다(lamda) : map,filter,reduce 활용

reduce 사용하기

- 이제 함수 f를 람다 표현식으로 만들어서 reduce에 넣어보자

```
>>> a = [1, 2, 3, 4, 5]
>>> from functools import reduce
>>> reduce(lambda x, y: x + y, a)
15
```