# 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(lamda)

```
lambda arguments: expression
```

- lambda: 람다 함수를 정의하는 키워드입니다
- arguments: 함수의 입력 인자, 콤마로 구분하여 여러 개를 넣을 수 있습니다.
- expression: 인자를 사용하는 표현식으로, 이 표현식의 결과가 함수의 반환 값이 됩니다.

람다 함수는 파이썬에서 간단한 함수를 한 줄의 코드로 작성할 수 있도록 해주는 익명 함수입니다.

# 람다(lamda)

**람다 함수의 특징**

- 이름을 지정하지 않고, 필요한 곳에 바로 작성하여 사용할 수 있습니다.
- 람다 함수는 작은 익명 함수입니다.
- 람다 함수는 임의의 수의 인수를 사용할 수 있지만 식은 하나만 사용할 수 있습니다.
- 식이 평가되고 반환됩니다.
- 람다 함수는 함수 개체가 필요한 곳이면 어디서나 사용할 수 있습니다.

# 람다(lamda)

**람다 함수의 특징**

•람다 식(또는 람다 함수)은 기본적으로 변수에 할당할 수 있는 코드 블록입니다,

•상위 함수를 지원하는 언어에서 인수로 전달되거나 함수 호출에서 반환됩니다.

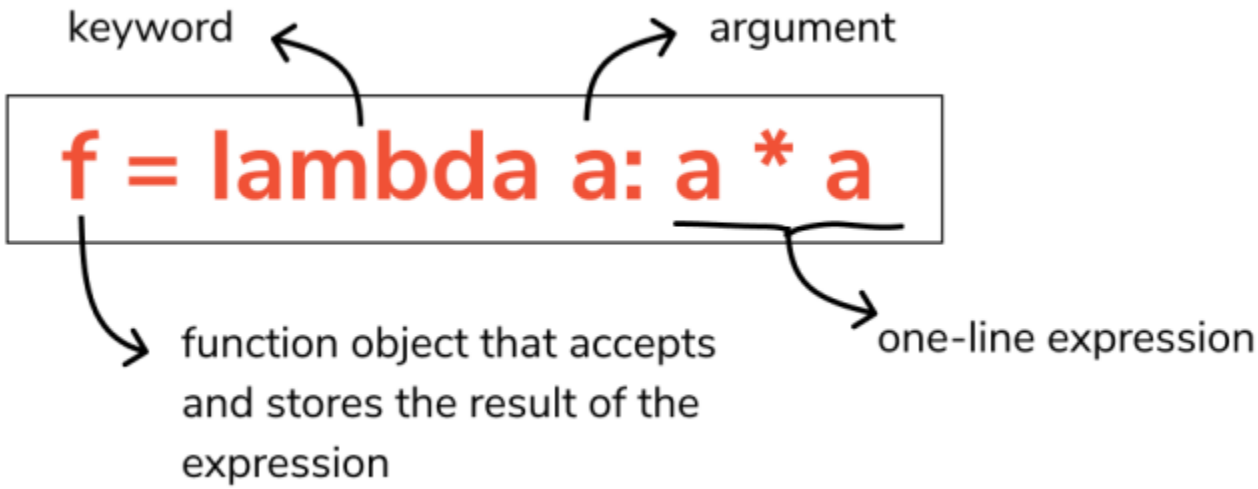
•코드를 간결하게 만들어 줍니다.

•람다 함수의 주요 역할은 익명으로 다른 함수 안에 서, 람다 함수를 사용할 때 시나리오에서 더 잘 설명됩니다 , 즉 고차 함수에 대한 인수로 활용될 수 있습니다

•람다 함수는 일반적으로 filter(), map() 또는 reduce() 같은 함수와 함께 사용되어 효율적인 데이터 처리를 돕습니다.

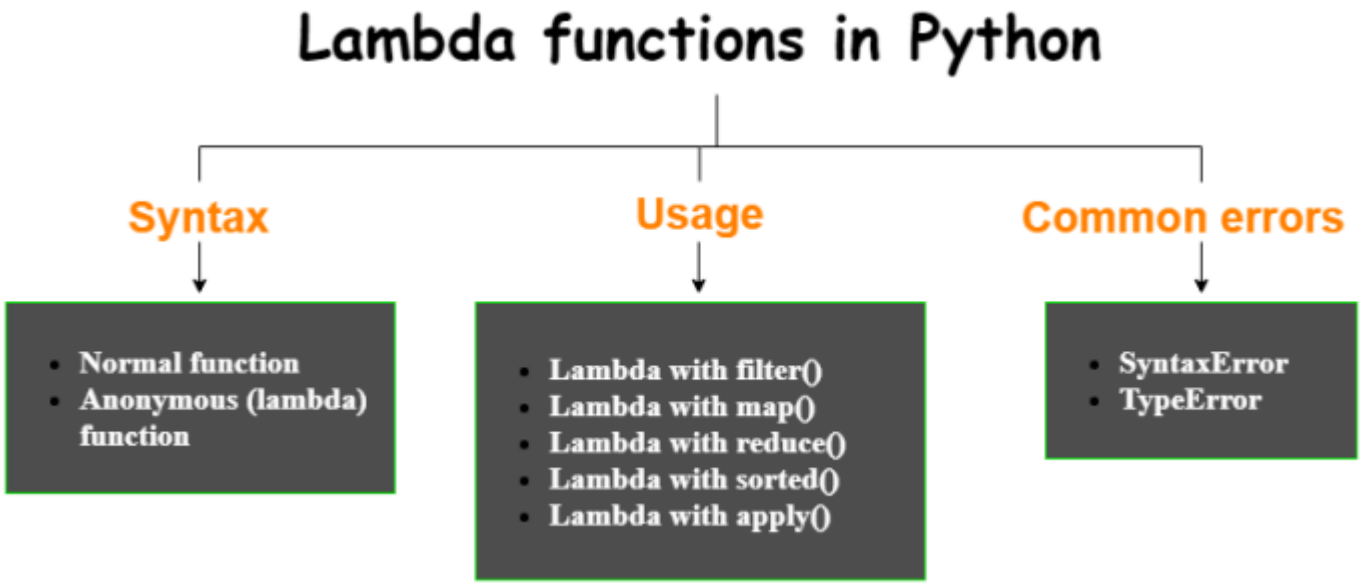
# 람다(lamda)

## 람다 함수의 특징



# 람다(lamda)

## 람다 함수의 특징



# 람다(lamda)

## 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

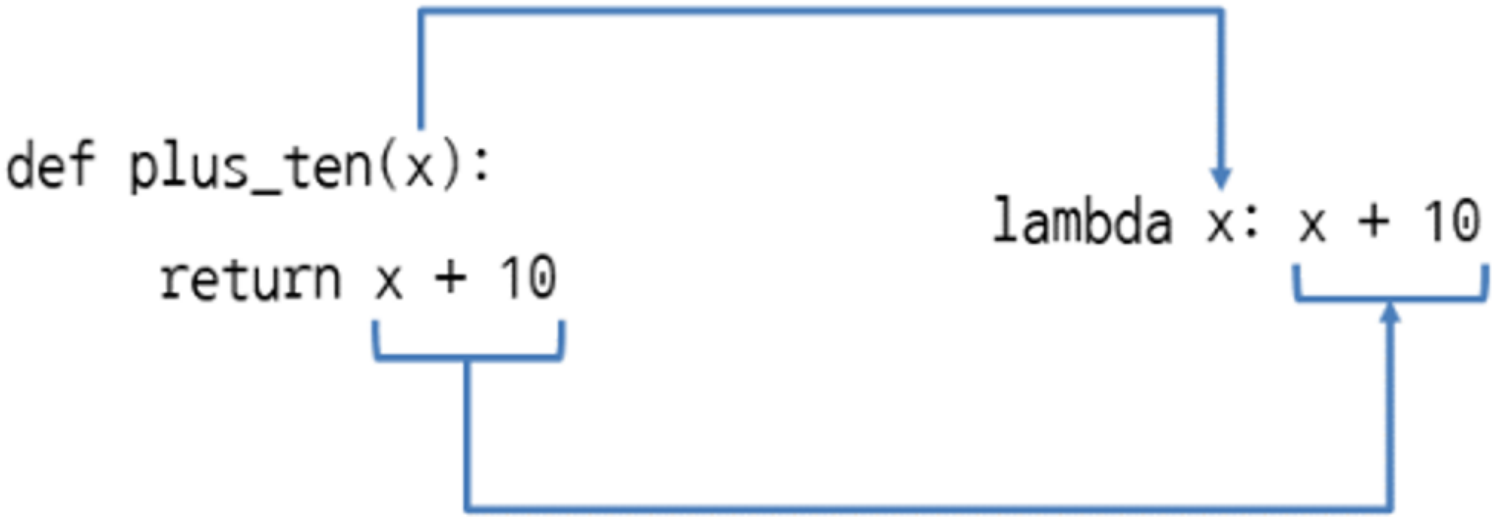
- 실행을 해보면 함수 객체가 나오는데, 이 상태로는 함수를 호출할 수 없음
- 람다 표현식은 이름이 없는 함수를 만들기 때문임
- 람다 표현식을 익명 함수(anonymous function)로 부르기도 함
- lambda로 만든 익명 함수를 호출하려면 다음과 같이 람다 표현식을 변수에

```
>>> 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)

```
# Define a function using 'def'
def f(x, y):
 return x + y

print('The sum of {} and {} is'.format(3.14, 2.718), f(3.14, 2.718))



# Define the same function using 'lambda
f = lambda x, y: x+y
print(f'The sum of pi number and euler number is {f(3.14, 2.718)}.')
```

# 람다(lamda)

```
# Calculate the volume of a cube using def and lambda functions
# def function
def cube_volume_def(a):
 return a*a*a
print(f'The volume of def function is {cube_volume_def(3.14)}.')


# lambda function
cube_volume_lambda = lambda a: a*a*a
print(f'The volume of lambda function is {cube_volume_lambda(3.14)}.')
```

# 람다(lamda)

## 함수식 안에서 람다 활용

```
def mult_table(n):
 return lambda x:x*n

n = int(input('Enter a number: '))

y = mult_table(n)

print(f'The entered number is {n}, which is a perfect number.')

for i in range(11):
 print(('%d x %d = %d' %(n, i, y(i))))
```

# 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
```

#Mul plica on of pi number and 12 in one line using lambda func on
print((lambda x:x*3.14) (12))
# Division
print((lambda x: x/3.14) (12))
# Addition
print((lambda x: x+3.14) (12))
# Subtract
print((lambda x: x-3.14) (12))
# Remainder
print((lambda x: x%3.14) (12))
# Floor division
print((lambda x: x//3.14) (12))
# Exponential
print((lambda x: x**3.14) (12))

# 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)

- 파이썬 함수 map()의 구문
- map(func, *iterables)
  - func: 각 요소에 적용할 함수입니다.
  - iterables: 함수를 적용할 데이터 집합입니다.

  - map() 함수는 iterable의 각 요소에 대해 function 함수를 적용한 결과를 새로운 iterator로 반환합니다. 이때, function 함수는 각 요소를 인자로 받아서 처리하며, 함수의 반환값이 새로운 iterator의 각 요소가 됨

# 람다(lamda):map

```
# standard
my_pets = ['alfred', 'tabitha', 'william', 'arla']
uppered_pets = []

for pet in my_pets:
    pet_ = pet.upper()
    uppered_pets.append(pet_)

print(uppered_pets)



# map
my_pets = ['alfred', 'tabitha', 'william', 'arla']

uppered_pets = list(map(str.upper, my_pets))

print(uppered_pets)
```

# 람다(lamda):map

2개 요소 람다 없이 함수로 인자 처리

```
def add(x, y):
    return x + y

numbers1 = [1, 2, 3, 4, 5]
numbers2 = [10, 20, 30, 40, 50]
added_numbers = map(add, numbers1, numbers2)

print(list(added_numbers))  # [11, 22, 33, 44, 55]
```

# 람다(lamda) : map,filter,reduce 활용

람다 표현식에 조건부 표현식 사용하기

- **lambda 매개변수들: 식1 if 조건식 else 식2**
- 다음은 map을 사용하여 리스트 a에서 3의 배수를 문자열로 변환함

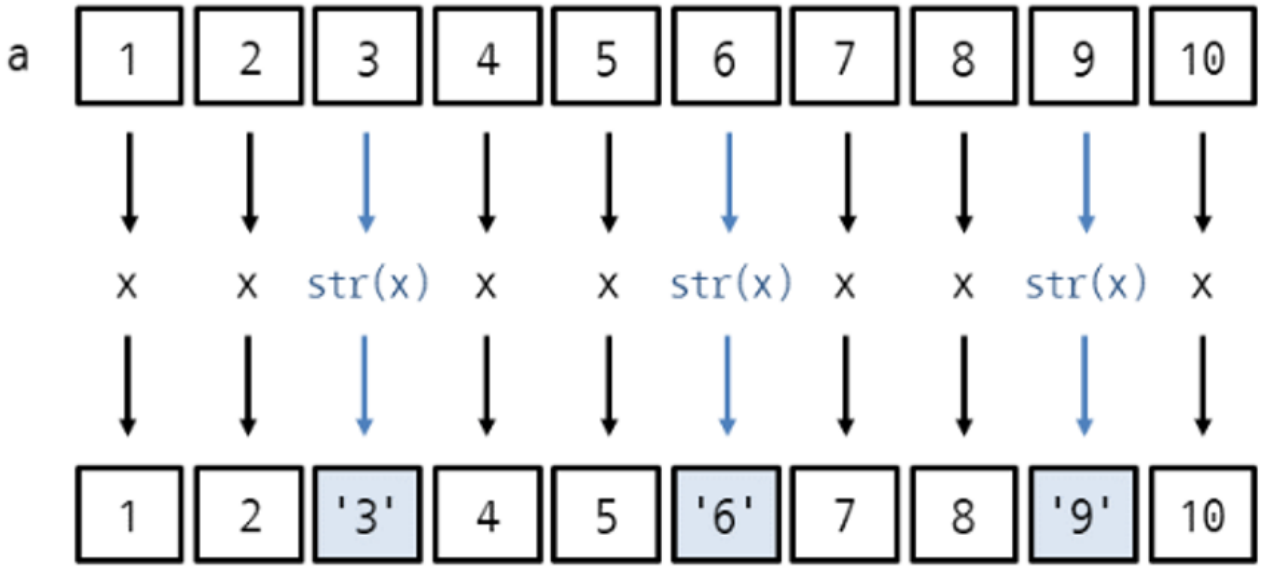
```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(map(lambda x: str(x) if x % 3 == 0 else x, a))
[1, 2, '3', 4, 5, '6', 7, 8, '9', 10]
```

- map은 리스트의 요소를 각각 처리하므로 lambda의 반환값도 요소라야 함
- 여기서는 요소가 3의 배수일 때는 str(x)로 요소를 문자열로 만들어서 반환했고, 3의 배수가 아닐 때는 x로 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 만듬

```
>>> 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) : filter

- map() 은 반복 가능한 객체의 각 요소를 함수에 전달하고, 함수를 통과한 모든 요소의 결과를 반환 하는 것이라면,
- Filter()는 먼저 함수가 부울 값(참 또는 거짓)을 반환하도록 요청.
- 다음 반복 가능한 객체의 각 요소를 함수에 전달한다.
- 논리에 거짓인 요소를 '필터링' 하여 제거하는 구조를 갖는다.

filter(func, iterable)

# 람다(lamda) : filter

특징:

1. 단일 이터러블 요구 사항
filter() 함수는 하나의 이터러블만 필요로 한다.

이것은 map()과의 주요 차이점 중 하나,

map()은 여러 이터러블을 받을 수 있다.

filter()의 간결함은 하나의 이터러블에 대해 함수를 적용하고,
그 함수가 True를 반환하는 요소만 추출하는 데에 있습니다.

# 람다(lamda) : filter

특징:

2. 함수는 불린 값을 반환해야 함
filter()에 전달되는 함수는 불린 값, 즉 True 또는 False를 반환해야 합니다(이 함수는 이터러블의 각 요소에 적용된다)

함수가 True를 반환하면, filter()는 이 요소를 출력에 포함시킵니다.
함수가 False를 반환하면, 요소는 제외됩니다.

제공된 함수가 불린이 아닌 값을 반환하는 경우:
예를 들어 정수, 문자열 또는 기타 유형을 반환하는 경우, 그 반환 값은 참 거짓으로 처리되어 예상치 못한 결과를 초래할 수 있습니다.

참 값에는 0이 아닌 숫자, 비어 있지 않은 문자열, 비어 있지 않은 리스트 등이 포함됩니다.
거짓 값에는 0, None, 빈 시퀀스 등이 포함됩니다.

# 람다(lamda) : filter

특징:

3. 필터링 메커니즘
"필터"라는 이름은 그 기능을 반영한다 , 즉 —입력 이터러블을 필터링한다.
각 요소를 func을 통해 전달한다는 것은 다음을 의미

- filter()는 이터러블의 각 요소를 반복합니다.
- 각 요소에 func를 적용합니다.
- func가 True를 반환하는 요소를 수집하여 반환합니다.

# 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

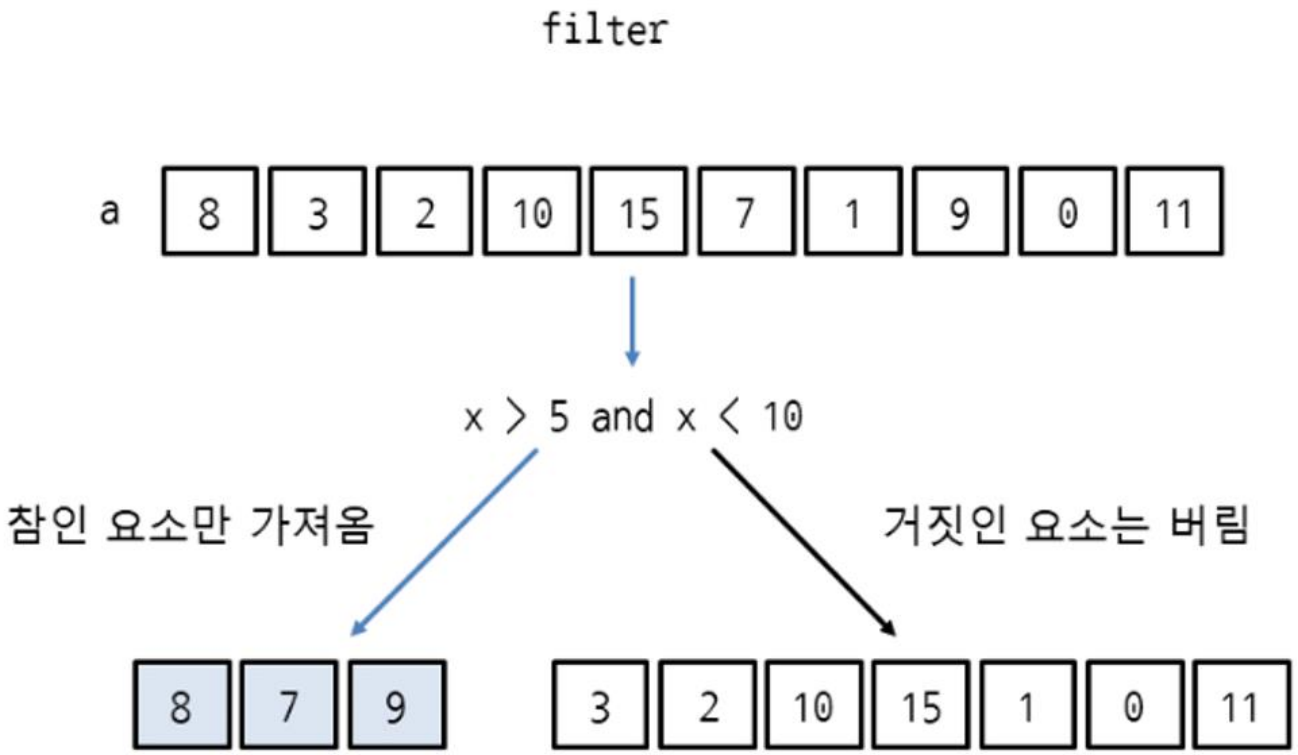
  - **filter( 함수, 반복가능한객체)**

```
>>> 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 < 10의 결과가 참인 요소만 가져오고 거짓인 요소는 버림

# 람다(lamda) : map,filter,reduce 활용

▼ **그림 32-3** filter 함수



# 람다(lamda) : map,filter,reduce 활용

```
# 숫자가 짝수인지 확인하는 함수 정의
def is_even(num):
    return num % 2 == 0

# filter를 사용하여 짝수만 추출
even_numbers = list(filter(is_even, numbers))

# 결과 출력
print(even_numbers)  # 출력: [2, 4, 6, 8, 10]
```

# 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

필터 활용 회문 분석:

```
dromes = ("구로구", "rewire", "madam", "freer", "마그마", "kiosk")

palindromes = list(filter(lambda word: word == word[::-1], dromes))

print(palindromes)
```

# 람다(lamda) : map,filter,reduce 활용

- reduce() 함수는 시퀀스의 요소들을 누적적으로 특정 함수에 적용하여 하나의 값으로 줄입니다. 이 함수는 functools 모듈에 포함되어 있습니다.

- 사용 방법:
  - reduce()는 주로 세 개의 인자를 받습니다:
    - 함수와 시퀀스, 그리고 선택적으로 초기값
  - 함수(Function): 두 개의 인자를 받아 하나의 값으로 결합하는 함수
  - 시퀀스(Sequence): 누적 적용될 시퀀스입니다.
  - 초기값(Initializer): 선택적으로 제공할 수 있는 초기 값입니다.

reduce(func, iterable[, initial])

# 람다(lamda) : map,filter,reduce 활용

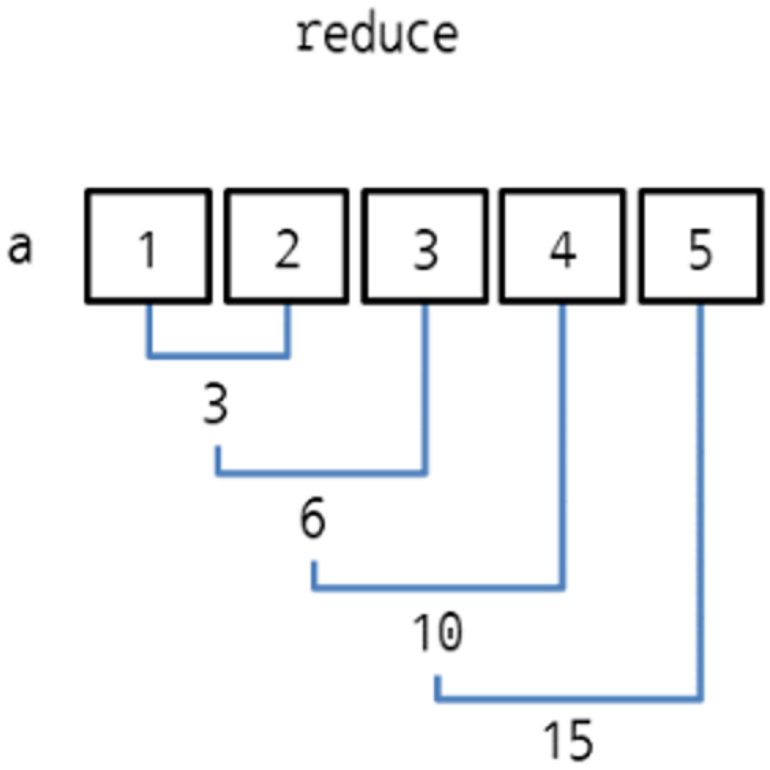
reduce 사용하기

- 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:초기값 적용

```
from functools import reduce

numbers = [3, 4, 6, 9, 34, 12]

def custom_sum(first, second):
    return first + second

result = reduce(custom_sum, numbers, 10)
print(result)
```

# 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
```

# 문제1

- **람다 리스트 정렬**
  - 주어진 사람들의 리스트를 각 사람의 나이에 따라 정렬하세요.
  - 람다 함수를 사용하여 정렬 기준을 제공합니다.

```
people = [
    {"name": "John", "age": 45},
    {"name": "Diana", "age": 32},
    {"name": "Tom", "age": 20}
]
```

```python
people = [
    {"name": "John", "age": 45},
    {"name": "Diana", "age": 32},
    {"name": "Tom", "age": 20}
]

# 나이에 따라 정렬
sorted_people = sorted(people, key=lambda x: x['age'])

# 결과 출력
print(sorted_people)  # 나이 순으로 정렬된 사람들의 리스트 출력
```

# 문제2

- 최대값 찾기
  - 주어진 숫자 리스트에서 최대값을 찾으세요. reduce()와 람다 함수를 사용하여 이 문제를 해결하세요.

```python
from functools import reduce

numbers = [5, 8, 6, 10, 9, 2]
max_number = reduce(lambda x, y: x if x > y else y, numbers)

# 결과 출력
print(max_number)  # 리스트에서 가장 큰 숫자 출력
```

# 문제3

- 조건에 맞는 요소 필터링
  - 주어진 숫자 리스트에서 5보다 큰 숫자만 포함하는 새 리스트를 만드세요. filter()와 람다 함수를 사용하여 이 문제를 해결하세요.

```python
numbers = [3, 5, 7, 10, 2, 6]
filtered_numbers = list(filter(lambda x: x > 5, numbers))

# 결과 출력
print(filtered_numbers)  # 5보다 큰 숫자들만 포함하는 리스트 출력
```

# 문제4

- 요소 변환
  - 주어진 문자열 리스트에서 각 문자열의 첫 글자만 대문자로 변환하세요. map()과 람다 함수를 사용하여 이 문제를 해결하세요.

```python
words = ["hello", "world", "python", "programming"]
capitalized_words = list(map(lambda word: word.capitalize(), words))

# 결과 출력
print(capitalized_words)  # 각 단어의 첫 글자를 대문자로 변환한 리스트 출력
```

# 문제5

- 복잡한 조건의 필터링
  - 주어진 제품 리스트에서 가격이 100보다 크고, 재고가 10개 이상인 제품만 추출하세요. filter()와 람다 함수를 사용하여 이 문제를 해결하세요.

  - products = [

{"name": "Product A", "price": 150, "stock": 5}, {"name": "Product B", "price": 120, "stock": 12}, {"name": "Product C", "price": 50, "stock": 20}, {"name": "Product D", "price": 200, "stock": 9}

]

```python
products = [
    {"name": "Product A", "price": 150, "stock": 5},
    {"name": "Product B", "price": 120, "stock": 12},
    {"name": "Product C", "price": 50, "stock": 20},
    {"name": "Product D", "price": 200, "stock": 9}
]

filtered_products = list(filter(lambda p: p['price'] > 100 and p['stock'] >= 10, products))

# 결과 출력
print(filtered_products)  # 조건에 맞는 제품들만 포함하는 리스트 출력
```

# 문제6

- 평균 점수 계산
  - 학생들의 점수 리스트에서 평균 점수를 계산하세요. reduce()와 람다 함수를 사용하여 이 문제를 해결하세요.

```python
from functools import reduce

scores = [88, 92, 78, 90, 84, 100, 95]
average_score = reduce(lambda x, y: x + y, scores) / len(scores)

# 결과 출력
print(f"The average score is: {average_score:.2f}")
```

# 문제7

- 특정 조건에 따른 문자열 필터링
  - 주어진 문자열 리스트에서 길이가 5 이상인 문자열만 추출하세요. filter()와 람다 함수를 사용하여 이 문제를 해결하세요.

  - ```
    words = ["apple", "pear", "banana", "cherry", "kiwi"]
    ```
    ```

```python
words = ["apple", "pear", "banana", "cherry", "kiwi"]
long_words = list(filter(lambda x: len(x) >= 5, words))

# 결과 출력
print(long_words)
```

# 문제8

- 각 요소의 제곱값 계산
  - 주어진 숫자 리스트의 각 요소에 대해 제곱값을 계산하세요. map()과 람다 함수를 사용하여 이 문제를 해결하세요.

```python
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x**2, numbers))

# 결과 출력
print(squared_numbers)
```

# 문제9

- 복수 필드에 따른 복합 필터링
  - 상품 리스트에서 가격이 100 이상이면서 재고가 50 이상인 상품만 추출하세요. filter()와 람다 함수를 사용하여 이 문제를 해결하세요.

```
products = [
    {"name": "Keyboard", "price": 150, "stock": 25},
    {"name": "Mouse", "price": 70, "stock": 90},
    {"name": "Monitor", "price": 200, "stock": 50},
    {"name": "USB cable", "price": 10, "stock": 150}
]
```

```python
products = [
    {"name": "Keyboard", "price": 150, "stock": 25},
    {"name": "Mouse", "price": 70, "stock": 90},
    {"name": "Monitor", "price": 200, "stock": 50},
    {"name": "USB cable", "price": 10, "stock": 150}
]
filtered_products = list(filter(lambda p: p['price'] >= 100 and p['stock'] >= 50, products))

# 결과 출력
print(filtered_products)
```

# 문제10

- 문자열 길이 변환
  - 주어진 문자열 리스트에서 각 문자열의 길이를 계산하여 새로운 리스트로 만드세요. map()과 람다 함수를 사용하여 이 문제를 해결하세요.

```python
words = ["hello", "world", "python", "code"]
lengths = list(map(lambda x: len(x), words))

# 결과 출력
print(lengths)
```

# 문제10

- 평균 점수와 평균 이상 점수 추출
  - 학생들의 점수 리스트에서 평균 점수를 계산하고, 평균 이상인 점수들만 추출하세요. reduce()와 filter()를 활용하여 이 문제를 해결하세요.

  - [88, 92, 78, 90, 84, 100, 95]

```python
from functools import reduce

scores = [88, 92, 78, 90, 84, 100, 95]
average = reduce(lambda x, y: x + y, scores) / len(scores)
above_average = list(filter(lambda x: x >= average, scores))

# 결과 출력
print(f"Average score: {average:.2f}")
print("Scores above average:", above_average)
```