

Day 8 - Call back, Lambda function

콜백함수 (callback function)

```
리 준비해준 함수입니다. 따라서 직접적으로 호출하지 않고 특정 조건이나  
def callback_func(func):  
    for i in range(5):  
        func()  
  
def print_hello():  
    print('안녕하세요! 파이썬!')  
  
print_hello()  
callback_func(print_hello)
```

- callback_func에서 print_hello의 주소를 가져와서 그 주소의 함수를 5번 실행시킨다.
- 함수의 매개변수로 함수를 넣을 수 있다.

람다함수 (Lambda function)

```
[9] def square(x):  
    return x**2  
  
print(square(5))  
  
25  
  
# lambda 매개변수: 표현식  
square = lambda x: x**2 # 람다는 무조건 결과가 리턴형  
print(square(5))  
  
25
```

- 결과가 무조건 리턴형이다.
- 변수에 저장이 가능하다.

```
(lambda x: x**2)(5)  
  
25
```

- 이런식으로 일회용으로 계산하는게 가능하다 (메모리를 차지 안한다)

sorted() 사용 풀이

```
sort_age(people[0])  
30  
[17] sorted_people = sorted(people, key=sort_age)  
print(sorted_people)  
[{'name': '김사과', 'age': 20}, {'name': '반하나', 'age': 25}, {'name': '오렌지', 'age': 30}]
```

- Key 안에 콜백함수를 넣을 수 있다.
- people 리스트 안에 있는 내용마다 다 실행시켜준다.
- output: age가 낮은순대로 정렬된다.
- 하지만 한번 정렬하는건데 메모리에 계속 살려놔야 한다.

Lambda식 사용 풀이:

```
sorted_people = sorted(people, key = lambda x: x['age'])  
print(sorted_people)  
[{'name': '김사과', 'age': 20}, {'name': '반하나', 'age': 25}, {'name': '오렌지', 'age': 30}]
```

- 콜백함수와 다르게 메모리에 남지 않는다.
- 같은 결과가 나온다.

filter 함수:

```
li = [2, 5, 7, 10, 15, 17, 20, 22, 25, 28]  
  
def even(n):  
    if n%2 == 0:  
        return True  
    else:  
        return False  
  
result = filter(even, li)
```

- 데이터를 내가 원하는것만 딱 골라서 자료구조에 담기위해서 쓴다
- 조건을 함수로 하면 한번만 쓸 거기 때문에 비효율적이다.
- 이터레이터 (반복에 쓸 수 있는 객체)
- output: [2, 10, 20, 22, 28]

```
✓ 0초 ▶ result = list(filter(lambda n: n%2 == 0, li))
print(result)
↔ [2, 10, 20, 22, 28]
```

- 람다식으로 조금 더 편하게 사용가능하다.

Map함수:

```
▶ # 함수를 모든 데이터에 적용하고 싶을때 맵 함수를 쓴다.
num = [1,2,3,4,5]
squared_num = list(map(lambda x: x**2, num)) # 1,2,3,4,5를 다 x**2에 대입시켜서 리턴해준다.
```

- 주어진 함수를 이터러블의 모든 항목에 적용하여 결과를 반환하는 이터레이터를 생성합니다.
- 주로 리스트나 순차적인 데이터타입에 사용

ex1)

```
▶ li1 = [1, 2, 3]
li2 = [4, 5, 6]
sum = list(map(lambda x, y: x+y, li1, li2)) # 2개의 파라미터를 넣을 수 있다.
print(sum)
↔ [5, 7, 9]
```

- Map을 이용해서 2개의 파라미터를 넣을 수 있다.

ex2)

```
↔ [5, 7, 9]
words = ['apple', 'banana', 'orange', 'cherry']
upper_words = list(map(lambda x: x.upper(), words))
```

- 하나하나 꺼내서 대문자로 바꾼다음 리스트로 만든다.
- ['APPLE', 'BANANA', 'CHERRY']