14강

데코레이터

14-1 데코레이터 개요

- 데코레이터
  - ✓ 데코레이터는 장식하다, 꾸미다라는 뜻의 decorate에 er(or)을 붙인 말이다.
    파이썬에서 데코레이터는 기존 클래스나 함수에 기능을 장식한다는 뜻으로 설명할 수 있다.
  - ✓ 클래스에서 특수한 메서드를 구성할 때 @staticmethod, @classmethod, @abstractmethod 등을 붙여서 사용하는데, 여기서 @로 시작하는 것들이 데코레이터이다.
  - ✓ 함수(메서드)를 장식하는 기능을 가지고 있다.

```
class AAA:

    @staticmethod
    def sum(a,b):
        return a + b
```

14-1 데코레이터 개요

- 데코레이터
  - ✓ 데코레이터는 함수를 수정하지 않은 상태에서 추가 기능을 구현할 때 사용한다.

```
def trace(func):
    def wrapper():
        print(func.__name__, '함수 시작')
        func()
        print(func.__name__, '함수 종료')

    return wrapper

def hello():
    print('hello')

def insa():
    print('insa')

hello = trace(hello)
hello()

insa = trace(insa)
insa()
```

[결과]

```
hello 함수 시작
hello
hello 함수 종료
insa 함수 시작
insa
insa 함수 종료
```

14-1 데코레이터 개요

- 데코레이터
  - ✓ @데코레이터 함수 이름을 이용할 수 있다.

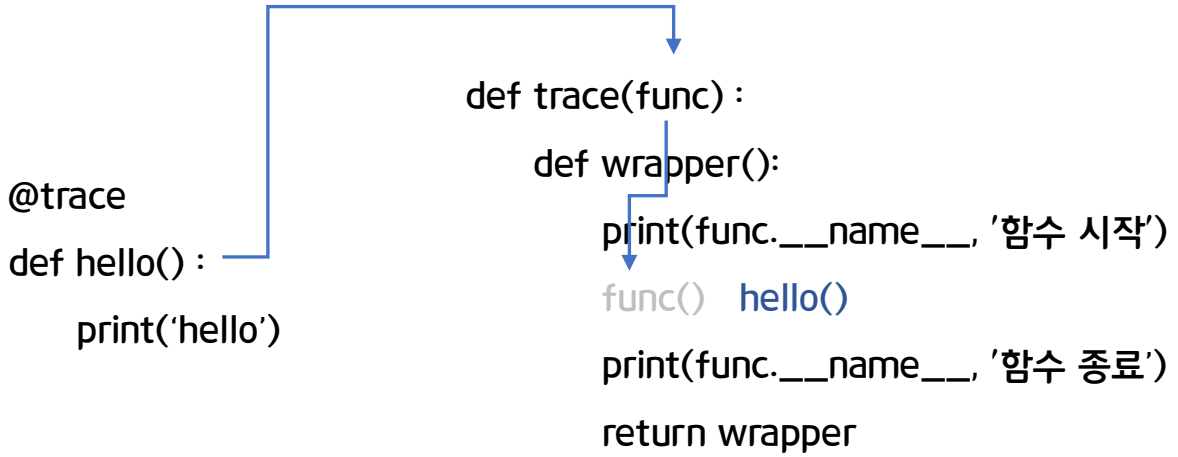
```
def trace(func):
    def wrapper():
        print(func.__name__, '함수 시작')
        func()
        print(func.__name__, '함수 종료')

    return wrapper

@trace
def hello():
    print('hello')

@trace
def insa():
    print('insa')

hello()
insa()
```




```
[결과]
hello 함수 시작
hello
hello 함수 종료
insa 함수 시작
insa
insa 함수 종료
```

14-2 데코레이터 활용(함수)

- 매개변수와 반환 처리
  - ✓ @데코레이터 함수 이름을 이용할 수 있다.
  - ✓ 매개변수가 고정되지 않은 함수를 사용할 때는 wrapper 함수를 가변 인수 함수로 만들어서 사용하면 된다.

```
def trace2(func):
    def wrapper(a, b):
        r = func(a, b)
        print('{} : a = {}, b = {} -> {}'.format(func.__name__, a, b, r))
        return r
    return wrapper

@trace2
def sum(a, b):
    return a + b
```

```
[결과]
sum : a = 40, b = 100 -> 140
140
```

```
def trace3(func):
    def wrapper(*args, **kwargs):
        r = func(*args, **kwargs)
        print('{0}(args={1}, kwargs={2}) -> {3}'.format(func.__name__, args, kwargs,r))
        return r
    return wrapper

@trace3
def get_max(*args):
    return max(args)

@trace3
def get_min(**kwargs):
    return min(kwargs.values())

print(get_max(10,40,29,50,100))
print(get_min(x=10, y=40, z=5))
```

```
[결과]
get_max(args=(10, 40, 29, 50, 100), kwargs={}) -> 100
100
get_min(args=(), kwargs={'x': 10, 'y': 40, 'z': 5}) -> 5
5
```

14-2 데코레이터 활용(클래스)

- 클래스
  - ✓ 클래스를 활용할 때는 인스턴스를 함수처럼 호출하게 해주는 __call__ 메서드를 구현해야 한다.

```
class CTrace:
    def __init__(self,func):
        self.func = func

    def __call__(self):
        print(self.func.__name__,'시작')
        self.func()
        print(self.func.__name__,'종료')
@CTrace
def insa():
    print('insa')
insa()
```

```
[결과]
insa 시작
insa
insa 종료
```

```
class CTrace2:
    def __init__(self, x):
        self.x = x

    def __call__(self, func):
        def wrapper(a, b):
            r  = func(a, b) * self.x
            print('{0}(a={1} b={2}) -> {3}'.format(func.__name__, a, b, r))
            return r
        return wrapper

@CTrace2(5)
def sum(a, b):
    return a + b

@CTrace2(10)
def subtract(a, b):
    return a - b

print(sum(20,40))
print(subtract(20,40))
```

```
[결과]
sum(a=20 b=40) -> 300
300
subtract(a=20 b=40) -> -200
-200
```