파이썬 프로그래밍

클래스와 객체

한국기술교육대학교
온라인평생교육원

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

• 클래스 내부에 메소드 선언 - def 키워드 사용
- 일반 함수와 다른 점은 첫번째 인수로 self 사용 (self라는 이름은 관례적)
  ° self: 인스턴스 객체 자신의 레퍼런스를 지니고 있음
  ° 각 인스턴스들은 self를 이용하여 자신의 이름 공간에 접근

```
class MyClass:
    def set(self, v):
        self.value = v
    def get(self):
        return self.value
```

- 클래스 안 메소듸 정의 시 def 키워드 활용
- 클래스 안에 존재하는 함수들 → 클래스의 메소드
- 클래스의 메소드는 첫 번째 인자에 self 넣음 → 인스턴스에 불려짐
- 첫 번째 인자에 self 들어간 것 → 인스턴스 메소드

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

- 인스턴스 객체를 통하여 메소드를 호출할 때 self 인자는 없다고 생각

```
c = MyClass() # 인스턴스 생성
c.set('egg')  # 메소드 set 호출
print c.get() # 메소드 get 호출
print c.value # 인스턴스 변수에 직접 접근
```

```
egg
egg
```

- c.set( ) → 인스턴스 c의 set 메소드 호출 → set은 인스턴스 메소드
- set이 인스턴스 메소드인 이유 → 첫 번째 인자가 self
- c 객체의 레퍼런스는 self로 copy됨 → c가 self가 됨
- set 정의 시 인자가 2개지만, 호출할 때는 1개만 줌
- c 인스턴스가 value 변수를 가짐
- c 인스턴스의 이름 공간에 value 생성 → value를 c 이름공간에서 호출

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

- 위 코드는 실제로 아래 코드와 동일함

```
c = MyClass() # 인스턴스 생성
MyClass.set(c, 'egg')
print MyClass.get(c)
print c.value
```

```
egg
egg
```

- self 자리에는 인스턴스가 들어가야 함
- 클래스 이름을 통해서 메소드 접근 → 직접 인스턴스 이름 삽입 필요

▣ 메소드의 정의와 호출

1. 일반 메소드의 정의와 호출

```
class Simple:
    pass


c = MyClass()
s1 = Simple()
MyClass.set(s1, 'egg')  # 다른 클래스의 인스턴스를 넣어주면 에러 발생
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-0660940f4507> in <module>()
      4 c = MyClass()
      5 s1 = Simple()
----> 6 MyClass.set(s1, 'egg')  # 다른 클래스의 인스턴스를 넣어주면 에러 발생

TypeError: unbound method set() must be called with MyClass instance as first
argument (got Simple instance instead)
```

- simple을 통해서도 인스턴스 생성이 가능
- self 자리에는 반드시 myclass 인스턴스만 올 수 있음
- c.set() → 이 순간의 set은 bound 메소드(특정 개체에 묶여 있음)
- MyClass.set() → 클래스 이름 통한 것은 unbound 메소드
- 특정 개체에 묶이지 않은 것
- class.메소드 호출 → unbound 메소드

```
• 메소드 호출 종류
 - Unbound method call: 클래스 객체를 이용한 메소드 호출
    * 예: MyClass.set(c, 'egg')
 - Bound method call: 인스턴스 객체를 통한 메소드 호출
    (self 인자는 호출받은 객체가 자동으로 할당)
    * 예: c.set('egg')
```

▣ 메소드의 정의와 호출

2. 클래스 내부에서의 메소드 호출

```
class MyClass:
    def set(self, v):
        self.value = v
    def incr(self):
        self.set(self.value + 1) # 내부 메소드 호출
    def get(self):
        return self.value

c = MyClass()
c.set(1)
print c.get()
print

c.incr()
print c.get()
```

```
1

2
```

- incr 메소드 내 다른 메소드를 호출하는 예제
- self 가 가지고 있는 value 값 + 1을 v에 넣음
- c 인스턴스가 self에 들어가고 self 자리엔 c가 존재
- incr() → increment의 약자

▣ 메소드의 정의와 호출

2. 클래스 내부에서의 메소드 호출

- 만약 위 코드에서 self.set(self.value + 1)를 set(self.value + 1)으로 바꾸면 set 함수를 정적 영역에서 찾는다.

```
def set(i):
    print "set function outside function - ", i

class MyClass:
    def set(self, v):
        self.value = v
    def incr(self):
        set(self.value + 1)     # 정적 영역에 존재하는 set 메소드 호출
    def get(self):
        return self.value

c = MyClass()
c.set(1)
print c.get()

print

c.incr()
print c.get()
```

```
1

set function outside function -  2
1
```

- set 앞에 self가 없으면 set 함수를 class 바깥에서 찾음
- 메소드 내 다른 메소드, 인스턴스 호출 시 반드시 self 사용

▣ 메소드의 정의와 호출

3. 정적 메소드

- 정적 메소드: 인스턴스 객체와 무관하게 클래스 이름 공간에 존재하는 메소드로서 클래스 이름을 이용하여 직접 호출할 수 있는 메소드
  - [주의] 해당 클래스의 인스턴스를 통해서도 호출 가능
- 장식자(Decorator) @staticmethod 활용

```
class D:
    @staticmethod
    def spam(x, y):             # self가 없다.
        print 'static method', x, y

D.spam(1,2) # 인스턴스 객체 없이 클래스에서 직접 호출

print
d = D()
d.spam(1,2) # 인스턴스 객체를 통해서도 호출 가능
```

```
static method 1 2

static method 1 2
```

- 클래스 이름공간에 존재하는 메소드 → 인스턴스 메소드도 마찬가지
- 클래스 내 메소드는 인스턴스를 통해 호출 & self 사용
- → 인스턴스 메소드
- 정적 메소드는 첫 번째 인자에 self 사용하지 않음
- 인스턴스를 통해서도 호출이 가능함
- 하지만, 인스턴스 이름공간에 작업 X → 클래스 이름공간 변수 조작
- spam 메소드 → 위에 장식자가 있으므로 static 메소드
- 인스턴스를 통해 호출 X → 첫 번째 인자에 self 필요 없음
- 인스턴스에 static 메소드 호출 가능

▣ 메소드의 정의와 호출

4. 클래스 메소드

- 클래스 메소드: 인스턴스 객체와 무관하게 클래스 이름 공간에 존재하는 메소드로서 클래스 이름을 이용하여 호출하며 첫 인수로 클래스 객체를 자동으로 받는 메소드
  - [주의] 해당 클래스의 인스턴스를 통해서도 호출 가능
- 장식자(Decorator) @classmethod 활용

```
class C:
    @classmethod
    def spam(cls, y):
        print cls, '->', y

print C

print
C.spam(5) # 첫번째 인수로 C가 잠재적으로 전달된다.

print
c = C()
c.spam(5) # 인스턴스 객체를 통해서도 호출 가능.
```

```
__main__.C

__main__.C -> 5

__main__.C -> 5
```

- 공통점 = 인스턴스 객체와 무관하게 클래스 이름 공간 존재
- 공통점 2 = 클래스 이름을 통해서 호출
- 클래스 메소드만 첫 번째 인자에 클래스 객체 자동으로 사용
- spam(cls, y) → C. spam(5) : y는 5, cls는 C 클래스 자동 삽입

▣ 메소드의 정의와 호출

4. 클래스 메소드

- 상속받은 서브 클래스를 통해 호출하면, 첫 인수에는 서브 클래스 객체가 자동으로 할당됨

```
class D(C):
    pass

print D.spam(3)

d = D()
print d.spam(3)
```

```
__main__.D -> 3
None
__main__.D -> 3
None
```

- D(C): → 클래스 D는 클래스 C를 상속 받음
- 상속 받으면 클래스 C가 가지고 있는 메소드 모두 가질 수 있음
- d라는 인스턴스를 통해서도 spam 호출 가능