Chapter 02. 파이썬 문법

8. 클래스

8.1 파이썬 클래스

1) 클래스는 새로운 이름 공간을 지원하는 단위

2) 데이터와 테이터를 변경하는 함수(메서드)를 같은 공간으로 작성

3) 클래스를 정의하는 것은 새로운 자료형을 만드는 것이고 인스턴스는 이 자료형의 객체를 생성하는 것이다.

4) 클래스 이름 공간과 이 인스턴스 의 이름공간은 각각 가지게 되며 유기적인 관계로 연결되어 있다.

[예제 class_def.py]

```
class MyString:
    pass

s = MyString()
print(type(s))
print(MyString.__bases__)

s2 = str()
print(type(s2))
print(str.__bases__)
```

8. 클래스

8.2 용어 정리

1) **클래스**: class 문으로 정의하며, 멤버와 메써드를 가지는 객체이다.

2) **클래스 객체**: 클래스와 같은 의미이지만 어떤 클래스를 구체적으로 가리킬 때 사용될 수 있다.

3) **클래스 인스턴스** : 클래스를 호출하여 만들어지는 객체

4) **클래스 인스턴스 객체** : 클래스 인스턴스와 같은 의미이다. 인스턴스 객체라 부르기도 한다.

5) **멤버** : 클래스가 갖는 변수

6) **메써드** : 클래스 내에 정의된 함수

7) **속성** : 멤버 + 메써드

8) **상위클래스** : 기반 클래스, 어떤 클래스의 상위에 있으며 여러 속성을 상속 해준다.

9) **하위클래스** : 파생 클래스, 상위 클래스로 부터 여러 속성을 상속 받는다.

8. 클래스

8.3 메서드의 정의와 호출

1) 메서드의 정의하는 방식은 함수를 정의하는 방법과 같다.

[예제 point.py]

```
class Point:

  def set_x(self, x):
      self.x = x

  def get_x(self):
      return self.x

  def set_y(self, y):
      self.y = y

  def get_y(self):
      return self.y
```

8. 클래스

2) 인스턴스 객체 생성과 메서드 호출

[예제 paint.py]

Bound Instance Method 호출

```
from point import Point

def main():
    bound_class_method()

def bound_class_method():
    p = Point()
    p.set_x(10)
    p.set_y(10)
    print(p.get_x(), p.get_y(), sep=',')

if __name__ == '__main__':
    main()
```

8. 클래스

Unbound class Method 호출(참고)

```
def main():
    unbound_class_method()

def unbound_class_method():
    p = Point()
    Point.set_x(p, 10)
    Point.set_y(p, 10)
    print(Point.get_x(p), Point.get_y(p), sep=',')

if __name__ == '__main__':
    main()
```

8. 클래스

3) 인스턴스 메서드 VS 정적 메서드 (static method) VS 클래스 메서드(class method)

인스턴스 메서드

메서드 내부에서 self(인스턴스 객체 자신)를 참조하기 위해 메서드 파라미터로 받게 된다.
첫 번째 파라미터를 받지 않게 되면, 인스턴스 객체에서의 호출은 실패한다.

[예제 point.py] – show() 메서드 추가

```
def show():
    print('점을 그렸습니다.')
```

[예제 paint.py]

```
def bound_instance_method():
  p = Point()
  p.set_x(10)
  p.set_y(10)
  p.show()
```

```
TypeError: show() takes 0 positional arguments but 1 was given
```

8. 클래스

따라서 인스턴스 객체 메서드는 객체 자신의 참조를 받을 수 있는 첫 파라미터가 있어야 한다.
보통 파라미터 변수 이름은 임의로 정할 수 있으나 관례적으로 self를 사용한다.

```
def show(self):
    print('점({0},{1})을 그렸습니다.'.format(self.x, self.y))
```

정적 메서드(staticmethod)와 클래스 메서드(classmethod)

인스턴스 메서드를 제외한 클래스 내에 정의된 메서드
인스턴스 객체의 멤버에 접근할 필요가 없는 메서드
첫 번째 인자로 인스턴스 객체 참조값을 받지 않는 클래스 내에 정의된 메서드라 할 수 있다.

[예제 point.py] – static_method(), class_method 메서드 추가

```
def static_method():
    return "static_method() 호출"

def class_method():
    return "class_method() 호출"
```

[예제 paint.py] – 테스트 함수 작성

```
def test_other_method():
    print(Point.static_method())
    print(Point.class_method())
```

8. 클래스

정적 메서드와 클래스 메서드의 차이점은 클래스 멤버에 접근과 관련이 있다.

[예제 point.py] – 클래스 멤버 정의와 static_method에서 접근

```
class Point:
    count_of_instance = 0
    . . .
    . . .
    def static_method():
        return count_of_instance;
```

[예제 paint.py] – 실행

```
def main():
    test_other_method()

def test_other_method():
    print(Point.static_method())
    print(Point.class_method())
```

```
NameError: name 'count_of_instance' is not defined
```

8. 클래스

클래스 메서드는 정적 메서드와 달리 클래스 멤버에 접근 하기 위해 클래스 객체 참조값을 첫 번째 파라미터로 받을 수 있는 메서드이다. 첫 번째 파라미터 변수의 이름은 임의로 정할 수 있으나 cls라는 이름을 사용한다.

[예제 point.py] – static_method와 class_method

```
class Point:
    count_of_instance = 0
    . . .
    . . .
   @staticmethod
    def static_method():
        return "static_method() 호출"

    @classmethod
    def class_method(cls):
        return cls.count_of_instanc
```

2.4 부터 지원하는 장식자(decorator)를 사용하면 클래스 메서드와 정적 메서드를 작성하는 데 복잡하지 않게 쉽게 작성할 수가 있다.

8.4 클래스 멤버와 인스턴스 멤버

멤버에는 클래스 멤버와 인스탄스 멤버 두 가지가 있다.

1) 클래스 멤버는 클래스 이름공간에 생성된다.

2) 인스턴스 멤버는 인스턴스 이름공간에 생성된다.

3) 클래스 멤버는 모든 인스턴스 객체들에서 공유된다.

4) 인스턴스 멤버는 각각의 인스턴스 객체에서만 참조된다.

[예제 point.py]

```
class Point:
    count_of_instance = 0

    def set_x(self, x):
        self.x = x
```

클래스 멤버 정의

인스턴스 멤버 정의

8. 클래스

[예제 paint.py] – 클래스 멤버와 인스턴스 멤버 접근

```
def test_member():
    p = Point()
    Point.set_x(p, 10)
    Point.set_y(p, 10)
    print('x={0}, y={1}, count_of_instance={2}'.format(p.x, p.y, p.count_of_instance))
```

인스턴스 객체에서 참조하는 멤버의 객체를 찾는 순서는 다음과 같다.

1. 인스탄스 멤버

2. 만일, 인스턴스 멥버가 존재하지 않으면 클래스 멤버를 찾게 된다.

8. 클래스

8.5 생성자와 소멸자

1) 파이썬에서는 __init__ 이름으로 생성자 를 작성할 수 있다.
2) 파이썬에서는 __del__ 이름으로 소멸자 를 작성할 수 있다.

[예제 point.py]

```
def __init__(self, x=0, y=0):
    self.x, self.y = x, y
    Point.count_of_instance += 1

def __del__(self):
    Point.count_of_instance -= 1
```

[예제 paint.py]

```
def test_constructor():
    p1 = Point(10, 20)
    print('x={0}, y={1}, count_of_instance={2}'.format(p1.x, p1.y, Point.get_count_of_instance()))
    p2 = Point(100, 200)
    print('x={0}, y={1}, count_of_instance={2}'.format(p2.x, p2.y, Point.get_count_of_instance()))
    del p1
    print('count_of_instance={0}'.format(Point.get_count_of_instance()))
    del p2
    print('count_of_instance={0}'.format(Point.get_count_of_instance()))
```

8. 클래스

8.6 __str__ 와 __repr__

1) __str__ 메서드

객체를 문자열로 반환하는 함수다.
문자열 형식은 __repr__ 메서드와 다르게 별 다른 제약이 없다.

[예제 point.py]

```
def __str__(self):
    return "Point({0}, {1})".format(self.x, self.y)
```

[예제 paint.py]

```
def test_to_string():
    p = Point()
    print(p)
```

8. 클래스

8.6 __str__ 와 __repr__

2) __repr__ 메서드

__str__ 메서드가 만들어 내는 문자열보다는 하나의 객체에 대한 유일한 문자열이어야 한다는 조건이 있다.
eval(...) 메서드를 통해 같은 객체로 재생이 가능한 문자열 표현이어야 한다.

[예제 point.py]

```
def __repr__(self):
    return "\"Point({0}, {1})\"".format(self.x, self.y)
```

[예제 paint.py]

```
def test_to_string():
    p = Point()
    print(p)
    print(repr(p))

    p2 = eval(repr(p))
    print(p2)
```

8. 클래스

[실습과제] 다음 조건을 만족하는 class Rect 작성하고 실습하기

1.인스턴스 멤버로 x1, y1, x2, y2를 가지고 있다.
2.생산자를 통해 4개의 인스턴스 멤버가 초기화 된다.
3. getter/setter 생략
4. __str__ , __repr__ 구현
5. draw(), area() 메서드 구현

```
def test_class_rect():
    r1 = Rect(10, 10, 50, 50)
    r2 = eval(repr(r1))

    print(r1, str(r1.area()), sep=':')
    print(r2, str(r2.area()), sep=':')

    r1.draw()
    r2.draw()
```

```
Rect(10, 10, 50, 50):1600
Rect(10, 10, 50, 50):1600
Rect(10, 10, 50, 50)를 그렸습니다
Rect(10, 10, 50, 50)를 그렸습니다
```

8.7 연산자 중복(Operator Oveloading)

연산자에 대해 클래스에 새로운 동작을 정의하는 것을 말한다.
파이썬에서는 사용하는 모든 연산에 대해 새롭게 정의할 수 있다.

1) 수치 연산자 오버로딩

[예제 mystring.py]

```
class MyString:

    def __init__(self, s):
        self.s = s

    def __truediv__(self, other):
        return self.s.split(other)

s = MyString('Python Programmer is Good')
t = s / ' '
print(type(t))
print(t)
```

8. 클래스

[예제 mystring.py]

오버로딩 되어 있지 않은 연산자를 사용하면 다음과 같은 오류가 발생한다.

```
print(s + " Job")
```

```
TypeError: unsupported operand type(s) for +: 'MyString' and 'str'
```

+ 연산자에 대한 오버로딩 메서드는 __add__(self, other) 이다.

```
class MyString:
. . .
. . .
def __add__(self, other):
        return self.s + other
```

다음과 같은 수치 연산자 메서드가 존재한다.

+:__add__ -:__sub__ *:__mul__ //:__floordiv__ %:__mod__ divmod():__divmod__

pow(),**:__pow__ <<:__lshift__ >>:__rshift__ &:__and__ ^:__xor__ |:__or__

8. 클래스

[실습과제]

MyString 클래스에 * 연산자 오버로딩을 해서 다음 결과가 나오도록 해보자

```
print(MyString("python") * 3)
```

```
pythonpythonpython
```

Point 클래스에 +, - 연산자를 각각 오버로딩 해서 다음 결과가 나올 수 있도록 해보자

```
def test_oerator_overloading():
    p1 = Point(100, 200)
    p2 = Point(50, 50)
    p3 = p1 + p2
    p4 = p1 - p2

    print(p3)
    print(p4)
```

```
Point(150, 250)
Point(50, 150)
```

8. 클래스

2) 역이항 연산자 오버로딩

[예제 mystring.py]

다음 예제를 실행시켜 보고 에러가 나는 이유를 생각해 보자.

```
print('Hello' + ' ' + MyString('World'))
```

```
TypeError: must be str, not MyString
```

MyString('....') + '...' 인 경우, __add__ 연산자가 호출되지만
'...' + MyString('....') 인 경우, __radd__ 연산자가 호출되기 때문에 에러가 발생한다.

해결하기 위해서는 __radd__ 함수를 구현해 주면 되지만 파라미터 순서에 유의해야 한다.

```
def __radd__(self, other):
    return other + self.s
```

이를 역이항 연산자 오버로딩 메서드라 한다. 다음과 같은 메서드를 지원하고 있다.

+:__radd__ -:__rsub__ *:__rmul__ //:__rfloordiv__ %:__rmod__ divmod():__rdivmod__

pow(),**:__rpow__ <<:__rlshift__ >>:__rrshift__ &:__rand__ ^:__rxor__ |:__ror__

8. 클래스

3) 확장 산술 연산자 오버로딩

다음과 같은 확장 산술 연산자 메서드를 제공하고 있다.

+=:__iadd__ -=:__isub__ *=:__imul__ /=: __idiv__ //=:__ifloordiv__ %=:__imod__
**=:__ipow__ <<=:__ilshift__ >>=:__irshift__ &=:__iand__ ^=:__ixor__ |=:__ior__

[예제 Point.py]

```
def __iadd__(self, other):
    return Point(self.x + other.x, self.y + other.y)

def __isub__(self, other):
    return Point(self.x - other.x, self.y - other.y)
```

[예제 Paint.py]

```
p3 += Point(-10, -10)
p4 -= Point(-10, -10)
print(p3)
print(p4)
```


8. 클래스

3) 확장 산술 연산자 오버로딩

다음과 같은 확장 산술 연산자 메서드를 제공하고 있다.

+=:__iadd__ -=:__isub__ *=:__imul__ /=: __idiv__ //=:__ifloordiv__ %=:__imod__
**=:__ipow__ <<=:__ilshift__ >>=:__irshift__ &=:__iand__ ^=:__ixor__ |=:__ior__

[예제 Point.py]

```
def __iadd__(self, other):
    return Point(self.x + other.x, self.y + other.y)

def __isub__(self, other):
    return Point(self.x - other.x, self.y - other.y)
```

[예제 Paint.py]

```
p3 += Point(-10, -10)
p4 -= Point(-10, -10)
print(p3)
print(p4)
```

8. 클래스

4) 수치 단항 연산자 오버로딩

다음과 같은 수치 단항 연산자 메서드를 제공하고 있다.

-:__neg__ +:__pos__ abs():__abs__ ~: __invert__

[예제 mystring.py]

```
class MyString:
    . . .
    . . .
    def __neg__(self):
        return self.s[::-1]

print(-MyString('python'))
```

8. 클래스

5) 비교 연산자 오버로딩

다음과 비교 연산자 메서드를 제공하고 있으며 결과는 다음과 같이 반환해야 한다.

<:__lt__ <=:__le__ >:__gt__ >=: __ge__ ==:__eq__ !=:__ne__

[예제 rect.py]

```
class Rect:
    def __eq__(self, other):
        return self.area() == other.area()

    def __lt__(self, other):
        return self.area() < other.area()

    def __gt__(self, other):
        return self.area() > other.area()
```

[예제 paint.py]

```
print(Rect(10, 20) == Rect(20, 10))
print(Rect(10, 10) > Rect(10, 5))
print(Rect(10, 20) < Rect(20, 10))
```

8. 클래스

6) 시퀀스 자료형의 연산자 오버로딩

다음과 비교 연산자 메서드를 제공하고 있으며 결과는 다음과 같이 반환해야 한다.

<:__lt__ <=:__le__ >:__gt__ >=: __ge__ ==:__eq__ !=:__ne__

[예제 rect.py]

```
class Rect:
    def __eq__(self, other):
        return self.area() == other.area()

    def __lt__(self, other):
        return self.area() < other.area()

    def __gt__(self, other):
        return self.area() > other.area()
```

[예제 paint.py]

```
print(Rect(10, 20) == Rect(20, 10))
print(Rect(10, 10) > Rect(10, 5))
print(Rect(10, 20) < Rect(20, 10))
```