Cf. 정적 메소드, 클래스 메소드

- **인스턴스 메소드, 정적 메소드, 클래스 메소드**

```
class CustomCalc:

    # instance method
    def add_instance_method(self, a,b):
        return a + b

    # classmethod
    @classmethod
    def add_class_method(cls, a, b):
        return a + b

    # staticmethod
    @staticmethod
    def add_static_method(a, b):
        return a + b


a=CustomCalc()
print(a.add_instance_method(3,5))
```

  - **실행결과**

```
8
```

  - 인스턴스 메소드 경우, 객체 할당하여 사용

- **인스턴스 메소드, 정적 메소드, 클래스 메소드 (cont')**

```
class CustomCalc:

    # instance method
    def add_instance_method(self, a,b):
        return a + b

    # classmethod
    @classmethod
    def add_class_method(cls, a, b):
        return a + b

    # staticmethod
    @staticmethod
    def add_static_method(a, b):
        return a + b

print(CustomCalc.add_class_method(3,5))

a=CustomCalc()
print(a.add_class_method(3,5))
```

  - **실행결과**

```
8
8
```

  - **classmethod도 객체할당으로 접근 가능**

- **인스턴스 메소드, 정적 메소드, 클래스 메소드 (cont')**

```
class CustomCalc:

    # instance method
    def add_instance_method(self, a,b):
        return a + b

    # classmethod
    @classmethod
    def add_class_method(cls, a, b):
        return a + b

    # staticmethod
    @staticmethod
    def add_static_method(a, b):
        return a + b


print(CustomCalc.add_static_method(3,5))

a=CustomCalc()
print(a.add_static_method(3,5))
```

  - **실행결과**

```
8
8
```

  - **staticmethod도 객체할당으로 접근 가능**

- **정적 메소드, 클래스 메소드 차이**

```
class Hello:
    t = '내가 상속해 줬어'

    @classmethod
    def calc(cls):
        return cls.t

class Hello_2(Hello):
    t = '나는 상속 받았어'

print(Hello_2.calc())
```

```
나는 상속 받았어
```

```
class Hello:
    t = '내가 상속해 줬어'

    @staticmethod
    def calc():
        return Hello.t

class hello_3(Hello):
    t = '나는 상속 받았어'

print(hello_3.calc())
```

```
내가 상속해 줬어
```

  - cls.t이 상속 시켜준 클래스에 있더라도 이것이 가리키는 것은 상속받은 클래스의 t 속성이다. cls는 상속 받은 클래스에서 먼저 찾는다.
  - staticmethod에서는 부모클래스의 클래스속성 값을 가져옴
  - classmethod에서는 cls인자를 활용하여 cls의 클래스속성을 가져옴
  - @classmethod : class 네임스페이스로 묶고 싶은 정적 메소드 이용
  - @staticmethod : 상속에서 사용되어 혼동을 초래할 여지가 없거나 조금이라도 더 간략하게 표현하고 싶을 때 이용

- **상속**

```
class Vehicle:
    def __init__(self, make, model, color,  price):
            self.__make = make          # 메이커
            self.model = model          # 모델
            self.color = color          # 자동차의 색상
            self.price = price          # 자동차의 가격

    def setMake(self, make):    # 설정자 메소드
            self.__make = make

    def getMake(self):          # 접근자 메소드
            return self.__make

        # 차량에 대한 정보를 문자열로 요약하여서 반환한다.
    def getDesc(self):
            return "차량 =("+str(self.__make)+","+₩
                                str(self.model)+","+₩
                                str(self.color)+","+₩
                                str(self.price)+")"

class Truck(Vehicle) :
    def __init__(self, make, model, color, price, payload):
            super().__init__(make, model, color, price)
            self.__payload=payload

    def setPayload(self, payload):      # 설정자 메소드
            self.__payload=payload

    def getPayload(self):               # 접근자 메소드
            return self.__payload
```

- **상속**

```
myTruck = Truck("Tesla", "Model S", "white", 10000, 2000)
print(myTruck.getDesc())
myTruck.setMake("benz")
myTruck.setPayload(3000)
print(myTruck.getDesc())
print('Payload :',myTruck.getPayload())


차량 =(Tesla,Model S,white,10000)
차량 =(benz,Model S,white,10000)
Payload : 3000
```