클래스

김지성 강사

객체 지향 프로그래밍(Object Oriented Programming)

- ✓ 객체지향 프로그래밍은 복잡한 문제를 작은 단위로 나누어 객체로 만들고, 객체를 조합해서 문제를 해결하는 것을 의미한다.
- ✓ 현실 세계의 복잡한 문제를 처리하는데 유용하며 기능을 개선하고 발전시킬 때도 해당 클래스만 수정하면 되므로 큰 프로젝트의 유지보수에도 매우 효율적이다.
- ✓ 객체가 가진 데이터를 **클래스의 속성(Attribute)**이라 부르고 객체가 갖는 기능을 **메서드(Method)**라고 부릅니다.

객체 지향 프로그래밍(Object Oriented Programming)

✓ 객체 vs 클래스 vs 인스턴스

- 객체 : 클래스가 실체화된 것.
- 클래스 : 객체를 만들어내기 위한 설계도로 이를 기반으로 여러 객체를 만들어낼 수 있다.
  - 속성(클래스에서 정의된 변수)과 메소드(클래스에서 정의된 함수)를 포함한다.
- 인스턴스 : 객체와 거의 동일한 의미로 사용된다.
  - 관계에 조금 더 포커싱을 맞춘 것으로 a = Car() 라고할 때, “a는 객체이고 a는 Car의 인스턴스다”라고 표현할 수 있다.

클래스(Class)

✓ 클래스는 사용자 정의 객체를 만들기 위한 설계도라고 생각하면 된다.

✓ 지금까지 사용해온 int, list, dict 등도 전부 클래스이다. 이 클래스로부터 인스턴스를 만들고 메소드를 사용한 것이다.

```
number = int(10)
print(type(number))

num_list = list(range(10))
print(type(num_list))

num_dict = dict(key="value", key2="value2")
print(type(num_dict))
```

```
<class 'int'>
<class 'list'>
<class 'dict'>
```

클래스(Class)

✓ 클래스를 정의할 때는 아래와 같이 정의하고 클래스의 **이름은 보통 파스칼 케이스**를 사용한다.

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

클래스(Class)

- ✓ 클래스를 정의할 때는 아래와 같이 정의하고 클래스의 이름은 보통 파스칼 케이스를 사용한다.
- ✓ 클래스를 정의할 때는 **class [클래스 이름]:** 으로 정의한다.
- ✓ 클래스를 인스턴스화 할 때는 **변수 = 클래스()로 인스턴스화**를 한다.

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

메소드(Method)

- ✓ **메서드(Method)는** 클래스 바디 안에서 정의되는 **함수**이다.
- ✓ 클래스의 인스턴스의 속성(attribute)으로서 호출되면, 그 메소드는 첫 번째 인자로 인스턴스 객체(self)를 받는다.
  - 이 때 첫 번째 인자를 설정 안 하면 에러가 발생. 이 첫 번째 인자를 'self'라고 사용한다.
  - 자기 자신을 참조하는 매개변수

```
class ClassName:
    def method_name(self):
        method_body
    class_body
    ...
```

메소드(Method)

✓ 메서드(Method)는 클래스 바디 안에서 정의되는 함수이다.

✓ 클래스의 인스턴스의 속성(attribute)으로서 호출되면, 그 메소드는 첫 번째 인자로 인스턴스 객체(self)를 받는다.

- 이 때 첫 번째 인자를 설정 안 하면 에러가 발생. 이 첫 번째 인자를 'self'라고 사용한다.

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

속성(Attribute)

- ✓ 클래스 속성(Attribute)을 만들 때는 **__init__메서드 안에서 self.속성에 값을 할당**하면 된다.
- ✓ **__init_메서드는** james = Person()처럼 클래스에 소괄호()를 붙여서 **인스턴스를 만들 때 호출되는 특별한 메소드(special method, magic method)이다**.
- ✓ __init__은 initialize의 줄임말로 **인스턴스(객체)를 초기화(메모리에 공간을 할당하고 값을 부여함)합니다**.
- ✓ 밑줄 두개(__double under, 던더)가 양 옆으로 붙어있는 메소드는 파이썬이 자동으로 호출하는 메소드입니다.
  - 스페셜 메서드(special method) 혹은 매직 메서드(magic method)라고 부른다. (던더 메소드로 불리기도 함)

속성(Attribute)

- ✓ 속성은 __init__메서드에서 만든다는 점과 self에 마침표(.)를 붙여 속성명을 붙이고 값을 할당하는 점을 반드시 기억해야 한다.
- ✓ 클래스 바디에서도 속성을 접근할 때 **self.속성** 과 같이 self에 마침표를 찍고 사용하면 된다.

```
class Person:
    def __init__(self):
        self.hello = '안녕하세요'

    def greeting(self):
        print(self.hello)

james = Person()
james.greeting()
```

```
안녕하세요
```

속성(Attribute)

✓ 인스턴스 속성 vs 클래스 속성

- 클래스 속성 : 모든 인스턴스에서 공유하는 속성
- 인스턴스 속성 : 개별 인스턴스들이 가지는 고유한 속성

```
class Person:
    bag = []

    def put_bag(self, stuff):
        self.bag.append(stuff)

james = Person()
james.put_bag('책')

maria = Person()
maria.put_bag('열쇠')

print(james.bag)
print(maria.bag)
```

```
['책', '열쇠']
['책', '열쇠']
```

클래스 속성

```
class Person:
    def __init__(self):
        self.bag = []

    def put_bag(self, stuff):
        self.bag.append(stuff)

james = Person()
james.put_bag('책')

maria = Person()
maria.put_bag('열쇠')

print(james.bag)
print(maria.bag)
```

```
['책']
['열쇠']
```

인스턴스 속성

속성(Attribute)

✓ 클래스 속성

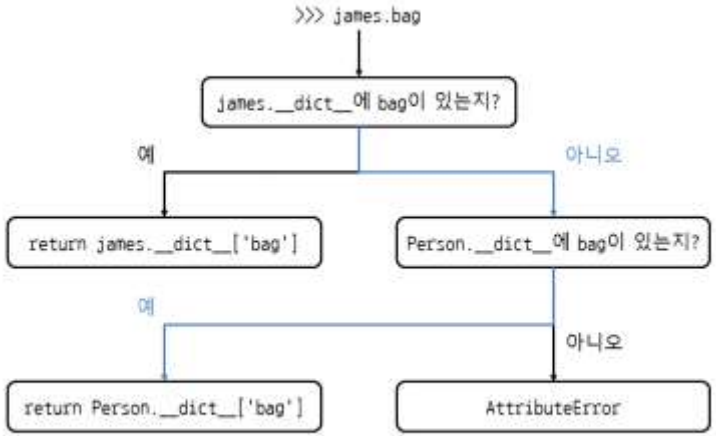
- 파이썬은 메소드 이름을 찾을 때 인스턴스->클래스 순으로 찾게된다. 따라서 james 인스턴스에 속성값이 없으므로 Person 클래스의 속성값을 가져온다.

```
james.__dict__
```

```
{}
```

```
Person.__dict__
```

```
mappingproxy({'__module__': '__main__',
              'bag': ['책', '열쇠'],
              'put_bag': <function __main__.Person.put_bag(self, stuff)>,
              '__dict__': <attribute '__dict__' of 'Person' objects>,
              '__weakref__': <attribute '__weakref__' of 'Person' objects>,
              '__doc__': None})
```



클래스 속성

속성(Attribute)

✓ 인스턴스 속성

- 인스턴스 속성을 보면 james 인스턴스에 속성값이 있는 것을 볼 수 있다.

```
james.__dict__
```

```
{'bag': ['책']}
```

```
Person.__dict__
```

```
mappingproxy({'__module__': '__main__',
              '__init__': <function __main__.Person.__init__(self)>,
              'put_bag': <function __main__.Person.put_bag(self, stuff)>,
              '__dict__': <attribute '__dict__' of 'Person' objects>,
              '__weakref__': <attribute '__weakref__' of 'Person' objects>,
              '__doc__': None})
```

인스턴스 속성

self의 의미

- ✓ self는 메서드가 호출된 **현재 객체(인스턴스)**를 참조한다.
- ✓ 클래스 내부에서 객체의 속성이나 메서드에 접근할 때 사용된다.
- ✓ 반드시 첫 번째 매개변수로 사용하며, 관례적으로 self를 사용한다.

- ✓ dog1.bark()를 호출하면 self는 dog1을 참조한다.
- ✓ dog2.bark()를 호출하면 self는 dog2을 참조한다
- ✓ 이처럼 여러 인스턴스를 만들 때 구분을 위한 역할로 사

```python
class Dog:
    def __init__(self, name):
        self.name = name  # 각 인스턴스의 이름을 저장

    def bark(self):
        print(f"{self.name}가 짖습니다!")

# 인스턴스 생성
dog1 = Dog("바둑이")
dog2 = Dog("흰둥이")

# 메소드 호출
dog1.bark()
dog2.bark()
```

```
바둑이가 짖습니다!
흰둥이가 짖습니다!
```

self의 의미

✓ self를 사용하지 않으면 name과 age는 일반 로컬 변수 취급을 받는다.

```python
class Dog:
    def __init__(self, name, age):
        self.name = name  # 인스턴스 변수
        self.age = age

    def bark(self):
        print(f"{self.name}가 짖습니다!")

# 객체 생성
dog1 = Dog("바둑이", 5)
dog2 = Dog("흰둥이", 3)

# 인스턴스 속성 및 메서드 호출
print(dog1.name)  # 출력: 바둑이
print(dog2.age)   # 출력: 3
dog1.bark()       # 출력: 바둑이가 짖습니다!
```

```
바둑이
3
바둑이가 짖습니다!
```

```python
class Dog:
    def __init__(name, age):  # self를 생략한 잘못된 코드
        name = name
        age = age

dog1 = Dog("바둑이", 5)
print(dog1.name)  # AttributeError: 'Dog' object has no attribute 'name'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 6
      3         name = name
      4         age = age
----> 6 dog1 = Dog("바둑이", 5)
      7 print(dog1.name)

TypeError: Dog.__init__() takes 2 positional arguments but 3 were given
```

속성(Attribute)

✓ 인스턴스를 생성할 때 속성 값을 할당하려면 다음 예제와 같이 __init___메서드에서 self 다음에 받을 값을 매개변수로 지정해야 한다.

✓ 그리고 매개변수를 self.속성에 할당한다.

✓ 그 후 인스턴스를 만들 때 추가된 매개변수만큼 인자를 넘겨줘야 한다.

```
class ClassName:
    def __init__(self, param1, param2):
        self.attr1 = param1
        self.attr2 = param2
```

속성(Attribute)

- ✓ 인스턴스를 만들 때 이름, 나이, 주소를 받는다.
- ✓ 인스턴스화를 할 때 다음과 같이 인자를 받는다.
  - Person(인자1,인자2,인자3)
- ✓ 그 후 인스턴스의 속성에 접근해서 출력해보면 값이 초기화된 것을 볼 수 있다.

```python
class Person:
    def __init__(self, name, age, address):
        self.hello = "안녕하세요."
        self.name = name
        self.age = age
        self.address = address

    def greeting(self):
        print(f"{self.hello} 제 이름은 {self.name}입니다.")

maria = Person("마리아", 20, "서울시 서초구 반포동")
maria.greeting()

print("이름:", maria.name)
print("나이:", maria.age)
print("주소:", maria.address)
```

```
안녕하세요. 제 이름은 마리아입니다.
이름: 마리아
나이: 20
주소: 서울시 서초구 반포동
```

비공개 속성(Private Attribute)

- ✓ 비공개 속성을 사용하면 인스턴스의 속성값의 수정이 불가능하다.
- ✓ 속성 앞에 던더(__)를 붙이면 비공개 속성이 된다.

```
class ClassName:
    def __init__(self, param1, param2):
        self.attr1 = param1
        self.__attr2 = param2
```

비공개 속성

비공개 속성(Private Attribute)

✓ Person 클래스의 wallet 속성을 비공개 속성으로 지정했기 때문에 wallet 속성을 변경하려고 하면 에러가 발생한다.

```
class Person:
    def __init__(self, name, age, address, wallet):
        self.name = name
        self.age = age
        self.address = address
        self.__wallet = wallet # 변수 앞에 __를 붙여서 비공개 속성으로 만듦

maria = Person('마리아', 20, '서울시 서초구 반포동', 10000)
maria.__wallet -= 10000 # 클래스 바깥에서 비공개 속성에 접근하면 에러 발생
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[35], line 9
      6                 self.__wallet = wallet # 변수 앞에 __를 붙여서 비공개 속성으로 만듦
      8 maria = Person('마리아', 20, '서울시 서초구 반포동', 10000)
----> 9 maria.__wallet -= 10000

AttributeError: 'Person' object has no attribute '__wallet'
```

비공개 속성(Private Attribute)

✓ __init__ 메소드로 초기화를 진행하는 것은 가능하기 때문에 비공개 속성으로 지정할 값은 수정이 이루어지지 않는 상수로 만드는 것이 좋다.

```
class Person:
    def __init__(self, name, age, address, wallet):
        self.name = name
        self.age = age
        self.address = address
        self.__wallet = wallet
    def pay(self, amount):
        if self.__wallet < amount:
            print('돈이 모자랍니다')
        self.__wallet -= amount
        print('이제 {0}원 남았네요.'.format(self.__wallet))

maria = Person('마리아', 20, '서울시 서초구 반포동', 10000)
maria.pay(3000)
```

```
이제 7000원 남았네요.
```

클래스 연습문제

✓ Person 클래스로 maria, james라는 인스턴스를 생성합니다

✓ maria와 james 두 사람의 나이, 이름, 주소를 받아와 객체에 저장하고 화면에 출력해보세요.

➢ **실행 결과:**

```
이름은 무엇인가요?: 마리아
나이는 무엇인가요?: 20
주소는 무엇인가요?: 서울시 강남구
이름은 무엇인가요?: 제임스
나이는 무엇인가요?: 21
주소는 무엇인가요?: 서울시 구로구
첫 번째 이름: 마리아
첫 번째 나이: 20
첫 번째 주소: 서울시 강남구
두 번째 이름: 제임스
두 번째 나이: 21
두 번째 주소: 서울시 구로구
```

클래스 연습문제

✓ 사용자로부터 체력, 마나, AP를 입력받아 옵니다.

✓ 주어진 코드에서 애니(Annie)클래스를 작성하고 티버(tibbers)스킬의 피해량이 출력되게 만들어보세요.

✓ 티버 피해량은 AP * 0.65 + 400 이며, AP(Ability Power, 주문력)는 마법 능력치를 뜻합니다.

➢ **실행 결과:**

```
체력, 마나, AP를 입력하세요: 511.68 334.0 298
티버: 피해량 593.7
```

➢ **실행 결과:**

```
체력, 마나, AP를 입력하세요: 1803.68 1184.0 645
티버: 피해량 819.25
```

데코레이터

김지성 강사

데코레이터(Decorator)

✓ 데코레이터라는 이름은 “장식하는 사람”을 의미한다. 이와 비슷하게 파이썬에서도 어떤 함수가 있을 때 **해당 함수를 직접 수정하지 않고 함수에 기능을 추가하고자 할 때 데코레이터를 사용**한다.

✓ 다음과 같은 함수가 있다고 했을 때 앞/뒤로 *을 추가로 출력하고자 할 때 hello라는 함수 자체를 수정하는 것이 아니라 hello라는 함수를 받아서 기능을 추가해서 사용하면 된다.

```
def hello():
    print("hello")
```

```
hello()
hello
```

데코레이터(Decorator)

✓ 데코레이터는 함수를 파라미터로 받는다. 기존 함수인 hello를 인자로 전달받고 새로운 함수인 deco_hello를 반환한다.

✓ **deco_hello = deco(hello) 과정을 바인딩** 이라고 표현한다.

```
def hello():
    print("hello 함수 : hello")

def deco(fn):
    def deco_hello():
        print("\n데코레이터")
        print("*" * 20)    # 기능 추가
        fn()               # 기존 함수 호출
        print("*" * 20)    # 기능 추가
    return deco_hello
```

```
deco_hello = deco(hello)

hello()
deco_hello()
```

```
hello 함수 : hello

데코레이터
********************
hello 함수 : hello
********************
```

데코레이터(Decorator)

- ✓ 파이썬에서는 이러한 작업을 편리하게 해주는 @ 기호를 제공한다.
- ✓ 단순히 어떤 함수(예: hello)에 기능을 추가하고자 한다면 해당 함수 위에 @데코레이터함수와 같이 적어주면 된다.

```
@deco
def hello2():
    print("hello 2")

hello2()
```

```
데코레이터
**********
hello 2
**********
```

클래스 메소드 vs 정적 메소드

✓ @classmethod

- **클래스 메소드를 정의할 때 사용하는 데코레이터**이다.
- 클래스 메소드는 인스턴스가 아닌 클래스 자체를 의미한다. **첫 번째 인자로 self 대신 cls를 받고 cls는 호출된 클래스 자체를 나타낸다**. 따라서 클래스 변수에 접근하거나 메소드를 호출할 때 유용하게 사용될 수 있다.
- 인스턴스를 생성하지 않고도 클래스 이름을 통해 호출할 수 있어서 클래스의 상태와 관련된 작업을 수행하는데 편리하다.

✓ @static_method

- **정적 메소드를 정의할 때 사용하는 데코레이터**이다.
- **정적 메소드는 self 혹은 cls를 받지 않고 인스턴스나 클래스 변수에 접근할 수 없다**.
- 특정 기능이 클래스에 논리적으로 속하지만 클래스 상태와는 무관할 때, 코드 가독성을 위해 보통 사용된다.
- 호출 시점에 인스턴스나 클래스의 상태를 필요로 하지 않고 고정된 작업을 할 때 사용한다.

클래스 메소드 vs 정적 메소드

✓ 클래스 메소드와 정적 메소드를 호출할 때 클래스 이름으로 접근하는 것이 관례이다.

```python
class MyClass:
    class_variable = 0  # 클래스 변수

    def __init__(self, value):
        self.instance_variable = value  # 인스턴스 변수
        MyClass.class_variable += value  # 생성 시 클래스 변수 증가

    # 일반 메소드
    def instance_method(self, value):
        self.instance_variable += value  # 인스턴스 변수에 접근
        MyClass.class_variable += value  # 클래스 변수에도 접근 가능
        print(f"instance_variable: {self.instance_variable}")
        print(f"class_variable: {MyClass.class_variable}")

    # 클래스 메소드
    @classmethod
    def class_method(cls, value):
        cls.class_variable += value  # 클래스 변수에 접근
        print(f"class_variable (from class method): {cls.class_variable}")

    # 정적 메소드
    @staticmethod
    def static_method():
        print("static_method called!")
```

```python
# 객체 생성
instance1 = MyClass(10)
instance2 = MyClass(20)

# 일반 메소드 호출
instance1.instance_method(5)
# Instance variable: 15
# Class variable: 35

# 클래스 메소드 호출
MyClass.class_method(5)
# Class variable (from class method): 40

# 정적 메소드 호출
MyClass.static_method()
# Static method called!
```

```
instance_variable: 15
class_variable: 35
class_variable (from class method): 40
static_method called!
```

클래스 메소드 vs 정적 메소드

✓ 클래스 메소드
- 클래스 변수를 읽거나 수정해야 할 때, 클래스 수준의 정보를 제공해야 할 때.
- 클래스의 인스턴스를 생성하는 데 사용하거나 초기화 과정을 캡슐화할 때 주로 사용

✓ 정적 메소드
- 클래스나 인스턴스 상태에 의존하지 않는 함수를 정의할 때
- 입력 데이터를 처리하고 반환하는 로직을 캡슐화할 때 사용가능하다.

특징	클래스 메서드(@classmethod)	정적 메서드(@staticmethod)
첫 번째 매개변수의 유무	cls (클래스를 나타냄)	없음
클래스 속성 접근 가능 여부	가능	불가능
주로 사용되는 목적	클래스 상태 변경, 대체 생성자	독립적인 유틸리티 기능 제공

클래스 연습문제

- ✓ 객체 생성 시 **차이름**, **배기량**, **생산년도**를 입력받고 **인스턴스 속성**으로 만들어 주세요.
- ✓ 차이름, 배기량, 생산년도는 직접 변경하지 못합니다.
- ✓ 차이름을 확인하는 함수와 변경하는 **함수**를 생성해보세요.
- ✓ 배기량에 따라 1000cc 보다 작으면 소형

  1000cc 이상 2000cc 이하 중형

  2000cc 보다 크면 대형을 출력하는 **인스턴스 함수**를 만드세요.
- ✓ 객체 생성 마다 등록된 차량 갯수를 기록하는 **클래스 속성**을 만들어 주세요.
- ✓ 총 등록된 차량 개수를 출력하는 **클래스 함수**를 만드세요.
- ✓ 차량의 엔진 크기를 입력하면 소/중/대형을 판별하는 **정적 메서드**를 만드세요.

상속

김지성 강사

상속

- ✓ 부모의 유산을 자식이 물려 받듯이 부모 클래스와 자식 클래스의 관계를 통해서 부모 클래스의 속성과 메소드를 자식 클래스가 물려받을 수 있다.
- ✓ 자식 클래스를 선언할 때는 소괄호로 부모클래스를 포함시킨다.
- ✓ 자식 클래스는 부모 클래스의 속성과 메소드를 적지 않아도 사용이 가능하다.

```
class 부모클래스:
    ...내용...

class 자식클래스(부모클래스):
    ...내용...
```

상속

✓ Korea클래스는 속성이 없지만 부모 클래스인 Country클래스의 속성과 메소드를 그대로 받고 있

```
class Country:
    """Super Class"""

    name = '국가명'
    population = '인구'
    capital = '수도'

    def show(self):
        print('국가 클래스의 메소드입니다.')


class Korea(Country):
    """Sub Class"""

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

    def show_name(self):
        print('국가 이름은 : ', self.name)
```

```
a = Korea('대한민국')
a.show()
a.show_name()
print(a.capital,':',a.name)
```

```
국가 클래스의 메소드입니다.
국가 이름은 :  대한민국
수도 : 대한민국
```

상속 - 메소드 오버라이딩(Method Overriding)

✓ 메소드 오버라이딩은 부모 클래스의 메소드를 자식 클래스에서 재정의 하는 것이다.

✓ Country에 있는 show() 메소드를 자식 클래스인 Korea에서 수정한 결과이다.

```
class Korea(Country):
    """Sub Class"""

    def __init__(self, name,population, capital):
        self.name = name
        self.population = population
        self.capital = capital

    def show(self):
        print(
            """
            국가의 이름은 {} 입니다.
            국가의 인구는 {} 입니다.
            국가의 수도는 {} 입니다.
            """.format(self.name, self.population, self.capital)
        )
```

```
a = Korea('대한민국', 50000000, '서울')
a.show()
```

```
국가의 이름은 대한민국 입니다.
국가의 인구는 50000000 입니다.
국가의 수도는 서울 입니다.
```

상속 - super()

✓ super() 키워드를 사용하면 자식 클래스 내에서도 부모 클래스를 호출할 수 있다.

```
class Country:
    """Super Class"""

    name = '국가명'
    population = '인구'
    capital = '수도'

    def show(self):
        print('국가 클래스의 메소드입니다.')

class Korea(Country):
    def __init__(self, name,population, capital):
        self.name = name
        self.population = population
        self.capital = capital

    def show(self):
        print(super().name, super().population, super().capital)
        super().show()
        print(
            """
            국가의 이름은 {} 입니다.
            국가의 인구는 {} 입니다.
            국가의 수도는 {} 입니다.
            """.format(self.name, self.population, self.capital)
        )
```

```
a = Korea('대한민국', 50000000, '서울')
a.show()
```

```
국가명 인구 수도
국가 클래스의 메소드입니다.

            국가의 이름은 대한민국 입니다.
            국가의 인구는 50000000 입니다.
            국가의 수도는 서울 입니다.
```

상속을 통한 정적 메소드, 클래스 메소드 비교

```
class Animal:

        type = "동물"

    @staticmethod
    def getType1():
        return Animal.type

    @classmethod
    def getType2(cls):
        return cls.type

    def __init__(self):
        self.hungry = 0
```

```
class Dog(Animal):

    type = "강아지"

    def __init__(self):
        super().__init__()

    def sound(self):
        print("멍멍")
```

```
Dog.getType1()
>> 동물

Dog.getType2()
>> 강아지
```

상속을 통한 정적 메소드, 클래스 메소드 비교

```
class Animal:

        type = “동물"

    @staticmethod
    def getType1():
        return Animal.type

    @classmethod
    def getType2(cls):
        return cls.type

    def __init__(self):
        self.hungry = 0
```

```
class Dog(Animal):

    type = “강아지"

    # 부모 클래스 상속
    def __init__(self):
       super().__init__()

    def sound(self):
        print(“멍멍”)
```

자식클래스가 부모 메서드 사용

```
Dog.getType1()
>> 동물

Dog.getType2()
>> 강아지
```

고정 메서드 : 자식 클래스가 부모 클래스의 변수를 다룸

클래스 메서드 : 클래스 레벨로 작업을 수행

cls매개변수를 통해 자식 클래스의 변수에 영향을 받음

클래스 연습문제

✓ [문제 1] Character 클래스를 만드세요.

- Character 클래스의 Health 속성에 200을 할당해주세요.
- Character 클래스에 Move() 메서드를 추가하고 메서드 사용시 Health 가 -10 이 됩니다.
- Character 클래스에 Rest() 메서드를 추가하고 메서드 사용시 Health 가 + 10 됩니다.
- 현재 Health를 알수있는 checkHealth() 메서드를 추가해주세요

클래스 연습문제

✓ [문제 2] Knight와 Healer 클래스를 만들어 주세요.

- Knight 와 Healer 클래스는 Charcter 클래스를 상속합니다.
- Knight 클래스는 Move() 사용시 Health 가 -5 더 소모
- Knight 클래스는 Attack() 추가하고 실행시 공격합니다를 출력해주세요
- Healer 클래스는 Mana속성을 추가해주세요 (생성시 100)
- Healer 클래스는 heal(character) 메서드를 추가하고 메서드는 character 들을 매개변수로 받습니다.
- Healer 클래스는 heal(character) 메소드 실행시 Mana가 -10되고 전달받은 character 객체의 rest() 메소드를 실행합니다.
- Healer 클래스는 현재 마나속성을 확인할수있는 checkMana() 메서드를 추가해주세요