OOP II

- 인스턴스 & 클래스 변수
- 인스턴스 & 클래스간의 이름공간
- 인스턴스 & 클래스 메서드(+ 스태틱 메서드)

인스턴스 & 클래스 변수

인스턴스 변수

- 인스턴스의 속성(attribute)
- 각 인스턴스들의 고유한 변수
- 메서드 정의에서 `self.변수명` 로 정의
- 인스턴스가 생성된 이후 `인스턴스.변수명` 로 접근 및 할당

활용법

```
class Person:

    def __init__(self, name):    # 인스턴스 메서드 (생성자)
        self.name = name         # 인스턴스 변수
```

In [ ]:
```
# 클래스를 정의 해봅시다.
```

In [ ]:
```
class Person:
    def __init__(self,name):  # 인스턴스 메서드 중에 특별한 ..  생성자 메서드
        self.name = name  # 인스턴스 변수를 정의 / 할당
                          # self : 인스턴스 자기자신을 뜻함
```

In [ ]:
```
# Person의 인스턴스를 각각의 name 변수를 출력해봅시다.
```

In [ ]:
```
iu = Person('IU')          # self옆에 name에 이게 들어가는 것!
```

In [ ]:
```
iu.name   # iu가 self인것?
```

In [ ]:
```
park = Person('Park')    # park이 self인것?
park.name
```

클래스 변수

- 클래스의 속성(attribute)
- 모든 인스턴스가 공유
- 클래스 선언 내부에서 정의

- 클래스.변수명 으로 접근 및 할당

활용법

```
class Person:
    species = 'human'

print(Person.species)
```

In [ ]:
```
# 클래스를 정의 해봅시다.
```

In [ ]:
```
class Person:
    # 클래스 내에 변수 지정 -> 클래스 변수
    species = '사람'
```

In [ ]:
```
Person.species      # 선언할 때 빼고 약간 불러올 때는 이런 느낌인듯 !!
```

In [ ]:
```
# 인스턴스를 만들어 봅시다.
```

In [ ]:
```
kim = Person()    # kim은 Person에 속하는 인스턴스이다? 인스턴스 생성!!
```

In [ ]:
```
# 클래스 변수에는 어떻게 접근할 수 있을까요?
```

In [ ]:
```
kim.species   # 클래스는 모든 인스턴스가 공유하는 것이어서
```

In [ ]:
```
# 클래스가 공유하는 변수라고 했는데, 객체에서 클래스 변수에 접근/재할당을 해봅시다.
```

In [ ]:
```
kim.species = '신'
```

In [ ]:
```
# 클래스 변수는 실제로 같은 값을 공유하는 걸까요? 값을 한 번 변경해 봅시다.
```

In [ ]:
```
kim.species
```

In [ ]:
```
Person.species     # 신일까 사람일까? 답은 사람!

# 즉 인스턴스 변수가 새롭게 할당되게 된다면 클래스 자체는 변하지 않는다!!!!
```

인스턴스 & 클래스간의 이름공간

이름공간 탐색 순서

- 클래스를 정의하면, 클래스가 생성됨과 동시에 이름 공간(namespace)이 생성된다.

- 인스턴스를 만들게 되면, 인스턴스 객체가 생성되고 해당되는 이름 공간이 생성된다.
- 인스턴스의 속성이 변경되면, 변경된 데이터를 인스턴스 객체 이름 공간에 저장한다.
- 즉, 인스턴스에서 특정한 속성에 접근하게 되면 **인스턴스 => 클래스** 순으로 탐색을 한다.

In [ ]:
```
class Person:

    def __init__(self, name):
        self.name = name # self.name이 인스턴스 변수

    def talk(self):
        print(self.name)
```

In [ ]:
```
# p1을 만들어 말해 봅시다.
```

In [ ]:
```
iu = Person('iu')
```

In [ ]:
```
iu.talk()
```

In [ ]:
```
# 아래의 코드일 때, name으로 접근하면 어떻게 될까요?
class Person:
    name = 'unknown'
    # 인스턴스 변수가 정의된 적 없음
    def talk(self):
        print(self.name)
```

In [ ]:
```
# 아래에서 p2를 만들어 확인해 봅시다.
```

In [ ]:
```
# 인스턴스 변수가 정의된 적 없음
jimin = Person()
jimin.talk() # 그래서 클래스 변수의 값을 출력한다.  # unknown 나온다!!!
```

- 즉, 인스턴스 변수에 없으면 클래스 변수꺼보여주고 인스턴스 변수에 있으면 인스턴스꺼 보여줌
- **class와 instance는 서로 다른 namespace를 가지고 있다.** 이해하기 !!!

In [ ]:
```
# p2의 이름을 직접 변경해 봅시다.
```

In [ ]:
```
%%html
<iframe width="800" height="500" frameborder="0" src="http://pythontutor.com/visualize.html#
```

메서드의 종류

인스턴스 메서드(instance method)

- 인스턴스가 사용할 메서드
- 클래스 내부에 정의되는 메서드의 기본값은 인스턴스 메서드
- 기본적으로는 인스턴스 메서드라는 것이구나!!
- **호출시, 첫번째 인자로 인스턴스 자기자신 `self` 이 전달됨**

활용법

```
class MyClass:
    def instance_method(self, arg1, arg2, ...):
        ...

my_instance = MyClass()
# 인스턴스 생성 후 메서드를 호출하면 자동으로 첫 번째 인자로 인스턴스(my_instance)가 들어갑니다.
my_instance.instance_method(.., ..)
```

클래스 메서드(class method)

- 클래스가 사용할 메서드
- `@classmethod` 데코레이터를 사용하여 정의
- **호출시, 첫 번째 인자로 클래스 `cls` 가 전달됨**

활용법

```
class MyClass:
    @classmethod
    def class_method(cls, arg1, arg2, ...):
        ...

# 자동으로 첫 번째 인자로 클래스(MyClass)가 들어간다.
MyClass.class_method(.., ..)
```

스태틱 메서드(static method)

- 클래스가 사용할 메서드
- `@staticmethod` 데코레이터를 사용하여 정의
- **호출시, 어떠한 인자도 전달되지 않음**

활용법

```
class MyClass:
    @staticmethod
    def static_method(arg1, arg2, ...):
        ...

# 아무런 일도 자동으로 일어나지 않는다.
MyClass.static_method(.., ..)
```

In [ ]:
```
# Myclass 클래스를 정의해 봅시다.
```

In [ ]:
```
class MyClass:
```

```
    # 인스턴스 메서드
    def instance_method(self):
        return self

    # 클래스 미서드
    @classmethod
    def class_method(cls):
        return cls

    # 스태틱 메서드
    @staticmethod
    def static_method(arg):
        return arg
```

In [ ]:
```
# MyClass 클래스의 인스턴스 생성해 봅시다.
```

In [ ]:
```
mc = MyClass()
```

In [ ]:
```
# 인스턴스 입장에서 확인해 봅시다.
```

In [ ]:
```
mc.instance_method()  # self가 리턴되었는데 아래처럼 보임
```

In [ ]:
```
print(id(mc),id(mc.instance_method()))  # id 동일  # 즉 리턴값이 동일한 것이다!!
```

In [ ]:
```
mc == mc.instance_method()          # 인스턴스랑 인스턴스 메서드의 self값은 같다!
```

In [ ]:
```
# 인스턴스는 인스턴스 메서드에 접근 가능합니다.
```

In [ ]:
```
# 인스턴스는 클래스 메서드에 접근 가능합니다. 다만, 하지는 맙시다
```

In [ ]:
```
mc.class_method()
```

In [ ]:
```
# Error => 첫 번째 인자가 없다. 위와 같이 자동으로 첫 번째 인자로 들어가는 것이 없습니다.
mc.static_method()
```

In [ ]:
```
# 인스턴스는 스태틱 메서드에 접근 가능합니다.
mc.static_method(1)          # 자동으로 넘겨주는 것은 없고 직접 넣어준 것이 전달된다.
```

비교 정리

인스턴스와 메서드

- 인스턴스는 3가지 메서드 모두에 접근할 수 있다.
- 하지만 인스턴스에서 클래스 메서드와 스태틱 메서드는 호출하지 않아야 한다. (가능하다 != 사용한다)

- 인스턴스가 할 행동은 모두 인스턴스 메서드로 한정 지어서 설계한다.

In [ ]:
```
# 위의 MyClass를 활용하여 클래스 입장에서 확인해 봅시다.
```

In [ ]:
```
# 클래스 메서드를 호출해 봅시다.
MyClass.class_method()
```

In [ ]:
```
# cls 내부적으로 클래스를 넘겨준다!
MyClass.class_method() == MyClass
```

In [ ]:
```

```

In [ ]:
```
# 스태틱 메서드를 호출해 봅시다.
```

In [ ]:
```
MyClass.static_method(3)
```

In [ ]:
```
# 인스턴스 메서드를 호출해 봅시다.
```

In [ ]:
```
MyClass.instance_method()
# instance가 호출했다면,
# mc.instance_method()일거고. 파이썬 내부적으로
# MyClass.instance_method(mc) 호출을 했을 것..
# 근데 클래스가 호출했다보니 인자가 없다고 뜸..
```

In [ ]:
```
MyClass.instance_method(mc)
```

클래스와 메서드

- 클래스 또한 3가지 메서드 모두에 접근할 수 있다.
- 하지만 클래스에서 인스턴스 메서드는 호출하지 않는다. (가능하다 != 사용한다)
- 클래스가 할 행동은 다음 원칙에 따라 설계한다. (클래스 메서드와 정적 메서드)
  - 클래스 자체(`cls`)와 그 속성에 접근할 필요가 있다면 **클래스 메서드**로 정의한다.
  - 클래스와 클래스 속성에 접근할 필요가 없다면 **정적 메서드**로 정의한다.
    - 정적 메서드는 `cls` , `self` 와 같이 묵시적인 첫번째 인자를 받지 않기 때문

[코드예시] Puppy

- Puppy 클래스의 속성에 접근하는 클래스 메서드를 생성해 봅시다.
- 클래스 변수 `population` 를 통해 개가 생길 때마다 증가 시키도록 하겠습니다.
- 개들은 각자의 이름(name)과 종(breed)을 갖고 있습니다.
- `bark()` 메서드를 호출하면 짖을 수 있습니다.

In [ ]:
```
# Puppy 클래스를 정의해 봅시다.
```

In [ ]:
```
class Puppy:
```

```
    population = 0

    def __init__(self, name, breed='멍멍이'):
        self.name = name
        self.breed = breed
        Puppy.population += 1

    def bark(self):
        # instance메서드?
        # instance변수의 값을 활용하는 함수니까 !!! 중요하당!!
        print(f'멍멍 {self.name}!!! 나는 {self.breed}')

    @staticmethod
    def info():
        # class메서드 하셔도 되지 않을까요?..
        # 상관없어요
        # 근데 class를 쓰나요?
        print('우리집 강아지입니다 >_<')

    @classmethod
    def get_population(cls):
        # 클래스 변수의 값을 쓸거! 함수에서
        # 그러니까 클래스 메서드로 정의를 하고,
        # cls로 넘겨주는 클래스를 활용해서 코드를 짠다.
        print(f'{cls.population}')


    def __del__(self):
        Puppy.population -= 1
```

In [ ]:
```
# Puppy 3 마리를 만들어보고,
```

In [ ]:
```
p1 = Puppy('초코', '푸들')
p2 = Puppy('댕댕이', '시츄')
```

In [ ]:
```
# 다양한 메서드를 호출해봅시다.
```

In [ ]:
```
Puppy.get_population()   # Puppy.get_population(Puppy) 의미와 같다
```

In [ ]:
```
Puppy.info()
```

In [ ]:
```
p3 = Puppy('꿀이', '말티즈')
```

In [ ]:
```
Puppy.population
```

In [ ]:
```
p4 = Puppy('잉')
```

In [ ]:
```
p4.breed
```

In [ ]:
```
# Puppy 에 어떠한 속성에도 접근하지 않는 스태틱 메서드를 만들어보겠습니다.
```

강의시간 추가설명

객체 : object ~의 것

- attribute(속성) / method(메서드)
- 클래스(사람) -> 클래스변수, 클래스메서드
- 인스턴스(혜인,우중,..) -> 인스턴스변수, 인스턴스메서드