

클래스와 인스턴스

인스턴스는 클래스를 통해 나온 object이다.
class는 모든 인스턴스에 해당하는 특징을 보유!
인스턴스는 class의 모든 특징 및 function을 보유한다.

OOP와 클래스, 그리고 예시

OOP와 클래스를 통해 비슷한 그룹을 묶을 수 있다.
예를 들어 동물 안에는 인간, 고양이, 토끼가 있다면
동물의 특징에 더해서 각자의 특징을 클래스를 통해 구현한다.

클래스는 클래스의 이름과 상속하는 클래스를 명시

```
class Animal(object)
# Animal class에서 instance 생성시 실행
# instance.age = age, inst.name = None
def __init__(self, age):
    self.age = age
    self.name = None
# Animal의 인스턴스가 생성될 때
myanimal = Animal(3)
# getter method는 외부에서 데이터 접근에 필요
# 설정된 inst.name/inst.age를 return
# 비권장되는 형식 a.age
def get_age(self):
    return self.age
def get_name(self):
    return self.name
# setter method는 외부에서 데이터 재설정
# 설정된 inst.name/inst.age를 새롭게 설정
# 비권장되는 형식 a.age = "infinite"
def set_age(self, newage):
    self.age = newage
# newname = ""는 default value로! 안전하게 생성한다
def set_name(self, newname = ""):
    self.name = newname
# setter method는 외부에서 데이터 재설정
# 설정된 inst.name/inst.age를 새롭게 설정
def __str__(self):
    return (...)
```

클래스의 상속

클래스는 다른 클래스의 기능을 계승할 수 있다 (상속)
상속에서는 새로운 정보나 동작을 추가 및 덮어쓸 수 있다.
(= 메서드 오버라이드) 단, Superclass (파생클래스)의 기능을
사용하려면 다시 Superclass의 __init__ 같은

ex) animal이라는 class 아래에서 생성된 클래스가 있다.

- 1) person -> student
- 2) cat
- 3) rabbit

다음 3개의 클래스는 animal의 기능을 모두 계승하고,
각각의 클래스에 맞는 동작 및 기능을 추가 및 편집 가능

```
# cat subclass는 animal의 기능을 계승
# __init__은 animal의 그것을 그대로 사용한다.
class Cat(Animal):
    def speak(self):
        print("meow")
```

```
def __str__(self):
    return "cat:..."
```

person subclass는 animal의 기능을 계승

```
class Person(Animal):
    def __init__(self, name, age):
        # animal의 __init__을 불러와서 name, age를
        # Animal.__init__(self, age)
        # set_name(name) method를 호출
        # 갑자기 animal의 것을 쓸 수 없으므로!
        self.set_name(name)
        self.friends = []
```

self.friend의 getter

```
def get_friends(self):
```

```
    return self.friends
```

self.friend의 setter

```
def add_friend(self, fname):
    if fname not in self.friends:
        self.friends.append(fname)
```

새로운 기능

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

새로운 기능

```
def age_diff(self, other):
    diff = self.age - other.age
    print(abs(diff), "year diff..")
# 원래 animal의 str의 기능을 override
def __str__(self):
    return "person..."
```

클래스 변수에 대해서

클래스 내부에서 정의된 변수는 인스턴스 전체가 공유
즉, 이를 통해 인스턴스를 만들면서 unique ID 부여 가능

Rabbit subclass는 Animal의 기능을 계승

```
class Rabbit(Animal):
    # 클래스 변수 tag가 선언
    tag = 1
    # 새로운 __init__을 선언한다
    def __init__(self, age, p1=None, p2=None):
        # Animal의 __init__ 호출 후에
        Animal.__init__(self, age)
        # p1, p2를 따로 선언하고
        self.p1 = parent1
        self.p2 = parent2
        # 인스턴스.rid에 tag (=1)를 부여하고
        self.rid = Rabbit.tag
        # 다음 인스턴스에 달리는 tag는 2로 increment
        Rabbit.tag += 1
```

```

(...)
# rid의 고유번호의 getter
def get_rid(self):
    return str(self.rid).zfill(3)
# parent1의 고유번호의 getter
def get_p1(self):
    return self.parent1
# parent2의 고유번호의 getter
def get_p2(self):
    return self.parent2

# +를 override한다
# 여기서는 self + other하면 새로운 Rabbit이 생성
# 이 insta.는 부모의 정보를 보유하고 있다.
def _add_(self, other):
    return Rabbit(0, self, other)

# ==, !=를 override한다.
# self와 other의 parent의 rid를 비교해서
def __eq__(self, other):
    # 두 부모님이 같은 경우
    parents_same =
        self.parent1.rid == other.parent1.rid
        and self.parent2.rid == other.parent2.rid
    # 두 부모님이 다른 경우
    parents_opposite =
        self.parent1.rid == other.parent2.rid
        and self.parent2.rid == other.parent1.rid

    # 하나라도 해당되면 두 개체의 부모님은 같음
    return parents_same or parents_opposite

```