OOP
(Object-Oriented Programming)

Python에서 모든 것은 객체이다.

```
>> dir(anything)
```

객체 지향 프로그래밍의 기본 단위인 클래스 만들기

클래스명 : 첫 문자를 대문자로 하는 CapWords 방식으로 명명
by “PEP8 Coding Convention”

클래스 멤버

- 메서드(method)
- 속성(property)
- 클래스 변수(class variable)
- 인스턴스 변수(instance variable)
- 초기자(initializer)
- 소멸자(destructor)

클래스 멤버

필드
데이터를 표현

메서드
행위를 표현

```python
class Rectangle:
    count = 0  # 클래스 변수

    # 초기자(initializer)
    def __init__(self, width, height):
        # self.* : 인스턴스변수
        self.width = width
        self.height = height
        Rectangle.count += 1

    # 메서드
    def calcArea(self):
        area = self.width * self.height
        return area
```

메서드

클래스의 행위를 표현하는 것 > 클래스 내의 함수
메서드의 첫번째 parameter는 항상 self를 갖음

클래스 변수

클래스 정의에서 메서드 밖에 존재하는 변수
"Rectangle.count"와 같이 접근

인스턴스 변수

클래스 내부에서 "self.변수명"으로 사용
클래스 외부에서 "객체변수.self.변수명"으로 사용

```python
def __init__(self, width, height):
    self.width = width
    self.height = height

    # private 변수 __area
    self.__area = width * height

# private 메서드
def __internalRun(self):
    pass
```

초기자

__init__()
인스턴스 변수를 초기화하거나 객체의 초기상태를 만듦

접근제한자

Python은 기본적으로 모든 멤버가 public
코딩 관례상 밑줄(_)을 사용하여 구분
_ > protected
__ > private

```python
class Rectangle:
    count = 0  # 클래스 변수

    def __init__(self, width, height):
        self.width = width
        self.height = height
        Rectangle.count += 1

    # 인스턴스 메서드
    def calcArea(self):
        area = self.width * self.height
        return area

    # 정적 메서드
    @staticmethod
    def isSquare(rectWidth, rectHeight):
        return rectWidth == rectHeight

    # 클래스 메서드
    @classmethod
    def printCount(cls):
        print(cls.count)


# 테스트
square = Rectangle.isSquare(5, 5)
print(square)   # True

rect1 = Rectangle(5, 5)
rect2 = Rectangle(2, 5)
rect1.printCount()  # 2
```

정적 메서드

메서드 앞에 @sataticmethod Decorator를
표시하여 정적 메서드임을 표시함
인스턴스 객체를 통하지 않고 클래스를 통해
직접 호출 할 수 있음. (self 인자 선언 하지않음)

클래스 메서드

메서드 앞에 @classmethod Decorator를
표시하여 클래스 메서드임을 표시함
암묵적으로 첫 인자를 클래스로 받아옴

객체 생성과 사용

```python
# 객체 생성
r = Rectangle(2, 3)

# 메서드 호출
area = r.calcArea()
print("area = ", area)

# 인스턴스 변수 엑세스
r.width = 10
print("width = ", r.width)

# 클래스 변수 엑세스
print(Rectangle.count)
print(r.count)
```

클래스를 사용하기 위해서는 먼저 객체(Object)를 생성

"객체 변수명" = 클래스명()
(만약, __init__() 함수가 있다면, parameter를 전달해야함)
>> r = Rectangle(2, 3)

인스턴스 메서드는 "객체변수.메서드명()"과 같이 호출
>> r.calcArea()

인스턴스 변수는 값을 읽거나 변경하는 일이 가능하다.
>> r.width = 10

객체 생성과 사용

```python
# 객체 생성
r = Rectangle(2, 3)

# 메서드 호출
area = r.calcArea()
print("area = ", area)

# 인스턴스 변수 엑세스
r.width = 10
print("width = ", r.width)

# 클래스 변수 엑세스
print(Rectangle.count)
print(r.count)
```

클래스 변수에 접근할 때
"클래스명.클래스변수명" 혹은 "객체명.클래스변수명" 둘다 허용

하지만,

```python
r = Rectangle(2, 3)

Rectangle.count = 50
r.count = 10    # count 인스턴스 변수가 새로 생성됨

print(r.count, Rectangle.count)  # 10  50 출력
```

값을 할당할 경우 다른 결과 출력하게 된다.
r.count = 10 > 객체에 10을 할당
Rectangle.count = 50 > 클래스에 50을 할당

>> 클래스 변수에 접근 할 때는 클래스명을 사용하는 것이 좋다.

클래스 상속과 다형성

```python
class Animal:
    def __init__(self, name):
        self.name = name
    def move(self):
        print("move")
    def speak(self):
        pass

class Dog (Animal):
    def speak(self):
        print("bark")

class Duck (Animal):
    def speak(self):
        print("quack")
```

클래스 상속

"자식클래스(부모클래스)"

호출

```python
dog = Dog("doggy") # 부모클래스의 생성자
n = dog.name # 부모클래스의 인스턴스변수
dog.move()   # 부모클래스의 메서드
dog.speak()  # 파생클래스의 멤버
```

다형성

```python
animals = [Dog('doggy'), Duck('duck')]

for a in animals:
    a.speak()
```