Ask Django

클래스

Object Oriented Programming (OOP) 배경

- 지원 언어 : Python, Ruby, C++, Swift, Objective-C, Java, C# 등
- 함수는 데이터의 **처리방법**을 구조화했을뿐, **데이터** 자체는 구조화하지 못했다.
- 큰 문제를 작게 쪼개는 것이 아니라, 먼저 작은 문제들을 해결할 수 있는 객체들을 만든 뒤, 이 객체들을 조합해서 큰 문제를 해결하는 상향식 (Bottom-up) 해결법을 도입

OOP 주요 특징

- Encapsulation (캡슐화) : 관련 특성/기능을 하나의 클래스에 결합
- Inheritance (상속) : 코드 재활용성 증대
  - 부모 클래스의 특성/기능을 자식 클래스가 물려받음
  - 자식 클래스는 물려받은 특성/기능을 활용/변경/무시/재정의
- Polymorphism (다형성, 위키피디아, 참고)
  - 다른 동작을 동일한 함수로 사용할 수 있도록 지원

Class #doc

- 단순히 **<사용자 정의 데이터 타입>**
  - **관련된** 다수의 변수와 함수의 묶음으로 구성
- ex) Circle 클래스, Rectangle 클래스, Person 클래스, Book 클래스
- Tip: 파이썬에서 함수명은 snake_case, 클래스명은 CamelCase

```
class circle:              # NEVER !!!
    pass

class favorite_article:    # NEVER !!!
    pass

class Circle:              # OK
    pass

class FavoriteArticle:     # OK
    pass
```

파이썬3 기본 클래스는 object를 상속받아도, 안 받아도 동일합니다.

- Python2 에서는 object를 상속받아야 New-style Class (파이썬 2.2 도입) #ref
- Python3 에서는 Old-style class는 제거되고, New-style class만 남았습니다.

```
# Python 2
class Python2OldStyleClass:
    pass

class Python2NewStyleClass(object):  # 파이썬2에서는 필히 object를 상속받아주세요.
    pass

# Python 3
class Python3NewStyleClass:          # 파이썬3에서는 상속받지 않아도 New-style Class
    pass

class Python3NewStyleClass(objet):
    pass
```

Circle 로직 구현 (Class를 만나기 전 세상)

데이터와 함수의 개별 처리

```
from math import sqrt

def get_area(circle):
    return circle['radius'] ** 2

def get_distance(circle1, circle2):
    return sqrt((circle1['x'] - circle2['x']) ** 2 + (circle1['y'] - circle2['y']) ** 2) \
        - (circle1['radius'] + circle2['radius'])

>>> circle1 = {'x': 10, 'y': 20, 'radius': 3}
>>> circle2 = {'x': 100, 'y': -40, 'radius': 10}

>>> get_area(circle1)
9

>>> get_area(circle2)
100

>>> get_distance(circle1, circle2)
95.16653826391968
```

커스텀 클래스 Circle - 훨씬 응집도 있는 처리

데이터와 함수의 통합

```
from math import sqrt

class Circle(object):
    def __init__(self, x, y, radius):  # 생성자 (Constructor)
        self.x = x
        self.y = y
        self.radius = radius

    def area(self):
        return self.radius ** 2

    def distance(self, other):
        return sqrt((self.x - other.x) ** 2 + (self.y - other.y) ** 2) - (self.radius + other.radius)

>>> circle1 = Circle(10, 20, 3)     # 객체 생성 시에, 생성자 함수가 호출되며 인자가 전달
>>> circle1.area()
9
>>> circle2 = Circle(100, -40, 10)
>>> circle2.area()
100
>>> circle1.distance(circle2)
95.16653826391968
```

커스텀 클래스 Person

```
class Person(object):
    def __init__(self, name, age, region):
        self.name = name
        self.age = age
        self.region = region

    def say_hello(self):
        print('안녕하세요. {}님. {}에서 오셨군요.'.format(self.name, self.region))

    def move_to(self, region):
        print('{}에서 {}(으)로 이사합니다.'.format(self.region, region))
        self.region = region

>>> tom = Person('Tom', 10, 'Seoul')           # Person타입의 변수 tom
>>> steve = Person('Steve', 10, 'Pusan')       # Person타입의 변수 steve

>>> gongyou = Person('공유', 37, '도깨비나라')
>>> gongyou.move_to('Canada')
'도깨비나라에서 Canada(으)로 이사합니다.'
>>> gongyou.say_hello()
'안녕하세요. 공유님. Canada에서 오셨군요.'
```

- 지정 클래스 타입의 변수 = 인스턴스 (Instance)
- 인스턴스 생성 문법
  - 함수를 호출하듯이, **클래스를 호출**하여 인스턴스 생성
- 클래스가 호출이 될 때, 클래스내 __init__ 함수가 자동 호출
  - 생성자(Constructor)라 하며, 해당 인스턴스를 초기화하는 역할
  - 클래스 호출 시에 넘겨진 인자는 모두 생성자의 인자로 넘겨짐.
- 인스턴스를 위한 함수/변수들을 **인스턴스 함수**, **인스턴스 변수**

클래스 변수와 인스턴스 변수

클래스 변수와 인스턴스 변수

- 클래스 변수 (Class Variable)
  - 클래스 공간에 저장
- 인스턴스 변수 (Instance Variable)
  - 각 인스턴스마다 개별 공간에 저장

우선 Java 코드

```java
import java.util.ArrayList;

public class Dog {
    ArrayList<String> tricks;    // Java에서의 인스턴스 변수 선언

    public Dog() {
        this.tricks = new ArrayList<String>();
    }

    public void add_trick(String trick) {
        this.tricks.add(trick);
    }

    public static void main(String[] args) {
        Dog dog1 = new Dog();
        dog1.add_trick("roll over");

        Dog dog2 = new Dog();
        dog2.add_trick("play dead");

        System.out.println(dog1.tricks);  // ["roll over"] 출력
        System.out.println(dog2.tricks);  // ["play dead"] 출력
    }
}
```

인스턴스 변수인 것 같지만, 클래스 변수

파이썬에서는 인스턴스 변수 선언 문법은 Java의 그것과 다릅니다.

```
class Dog:
    tricks = []  # 이는 인스턴스 변수가 아니라, 클래스 변수입니다.

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> dog1 = Dog()
>>> dog1.add_trick('roll over')

>>> dog2 = Dog()
>>> dog2.add_trick('play dead')

>>> dog1.tricks    # ['roll over', 'play dead'] 출력 - ???????
>>> dog2.tricks    # ['roll over', 'play dead'] 출력
```

올바른 인스턴스 변수 선언

인스턴스 함수 내에서 인스턴스 변수를 선언합니다.

```
class Dog:
    def __init__(self):
        self.tricks = []  # 이것이 인스턴스 변수입니다.

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> dog1 = Dog()
>>> dog1.add_trick('roll over')

>>> dog2 = Dog()
>>> dog2.add_trick('play dead')

>>> dog1.tricks  # ['roll over'] : 바르게 출력
>>> dog2.tricks  # ['play dead'] : 바르게 출력
```

Data Hiding, Name Mangling

데이터 은닉과 이름 장식

- mangle : [동사] 짓이기다. 심하게 훼손하다.
- 파이썬에서는 접근 제한자 (public, private, protected) 미지원
- 언더스코어 2개 (__) 로 시작하는 이름에 한하여 이름을 변경 (Name Mangling) 기법을 제공
  - 인스턴스 함수 내에서는 이름 그대로 접근
  - 클래스 밖에서는 "_클래스명변수명" 으로 접근

```
class Person:
    def __init__(self, name):
        self.__name = name

    def say_hello(self):
        print('안녕 {}.'.format(self.__name))  # 인스턴스 함수 내에서는 이름 그대로 접근

>>> tom = Person('tom')
>>> tom.__name
AttributeError: 'Person' object has no attribute '__name'
>>> tom._Person__name    # 외부에서는 맹글링된 이름으로 접근
'tom'
>>> tom.say_hello()
'안녕 tom.'
```

Life is short,
use Python3/Django.