Chapter 10 : 객체 지향 프로그래밍 (Objective Oriented Programming), OOP

-객체: 실생활에서 일종의 물건, 속성과 행동을 가짐
-OOP는 객체 개념을 프로그램으로 표현, 속성은 variable, 행동은 함수로 표현됨
-설계도에 해당하는 클래스(class)와 인스턴스,객체(instance or object)로 나뉜다.

Objects in Python

*class 선언

cf) snake case: 띄워쓰기 부분에 "_"를 추가, 파이썬 함수/변수명에 사용
Camel Case: 띄워쓰기 부분에 대문자, class 명에 사용

Attribute 추가는 __init__ , self와 함께!
__init__은 객체 초기화 예약 함수

```
class SoccerPlayer(object):
    def __init__(self, name, position, back_number)
        self.name = name
        self.position = position
        self.back_number = back_number
```

Self는 인스턴스와 관련이 있다. class양식에 맞추어서 인스턴스를 선언하면 해당하는 값이 할당이 된다.

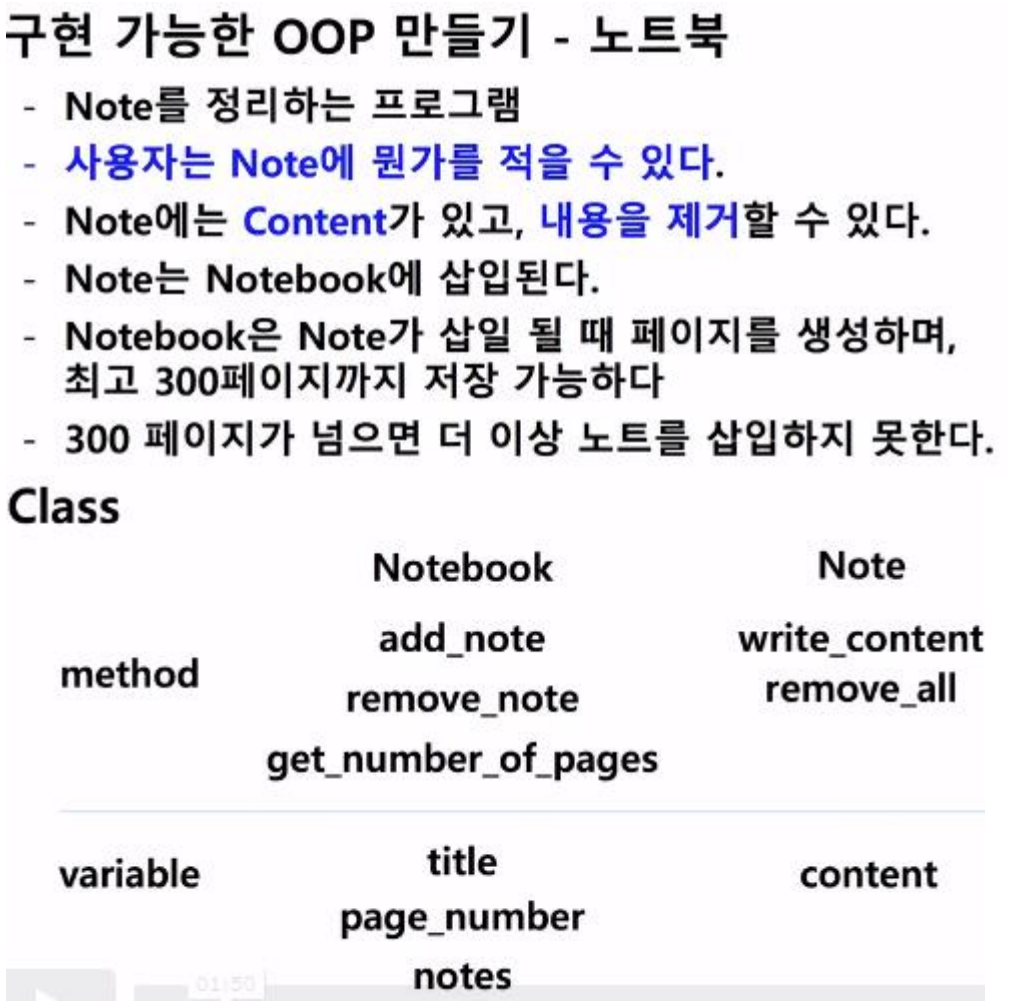
Class 구현하기 in Python - Function

Function(Action) 추가는 기존 함수와 같으나,
반드시 self를 추가해야만 class 함수로 인정됨

```
class SoccerPlayer(object):
    def change_back_number(self, new_number):
        print("선수의 등번호를 변경합니다 :
              From %d to %d" % ₩
             (self.back_number, new_number))
        self.back_number = new_number
```

- def __str__(self) : print문을 실행하였을 때, 값을 산출해주는 부분이다. 함수의 값은 반드시 "str" Type으로 지정해야한다.
- Named tulple : 속성값만 사용할 때, Class: 속성값과 함수 값을 함께 사용할 때

LAB: Note and Notebook



Object Oriented Programming의 특징

- 실제생활을 모델링 하기 위해선 여러 특징이 있다.(Inheritance, Polymorphism, Visibility....)

상속(Inheritance)

-부모클래스로부터 속성과 Method를 물려받는 자식 클래스를 생성하는 것

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

class Korean(Person):
    pass

first_korean = Korean("Sungchul", 35)
print(first_korean.name)  Sungchul
```

* korean에 별도의 명령어가 없지만 Person의 명령어를 상속받아서 동일한 결과를 산출한다.

다형성(Polymorphism)

-같은 이름 메소드의 내부 로직을 다르게 작성

[def get_some_data(**name**), def get_some_data(**name_date**)]

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

    def talk(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'
```

가시성(Visibility)

-객체의 정보를 볼 수 있는 레벨을 조절하는 것

-누구나 객체 안에 모든 변수를 볼 필요가 없음

```
class Inventory(object):
    def __init__(self):
        self.__items = []

    def add_new_item(self, product):
        if type(product) == product:
            self.__items.append(product)
            print("new item added")
        else:
            raise ValueError("Invalid Item")

    def get_number_of_items(self):
        return len(self.__items)

my_inventory = Inventory()
print(my_inventory.__items)
```

*** __ 표시(Private 변수)는 외부 클래스에서 마음대로 접근 할 수 없다. 클래스 내부에서만 접근 가능

```
@property
def items(self):
    return self.__items

print(my_inventory.items) # 접근 가능
```

*** @property를 사용하여 접근 가능하게 만들어준다. 함수를 변수처럼 호출

__items의 메모리 주소를 가져온다.