

9강

클래스

9-1 클래스 기본

- 객체 지향 프로그래밍

- ✓ 객체 지향 프로그래밍은 함수처럼 어떤 단순 기능만 사용하는 것이 아니라, 기능 여러 개를 묶고 여러 데이터를 하나의 클래스에 넣어 다른 프로그래머가 재사용할 수 있도록 하는 것이다.

- 클래스

- ✓ 클래스는 객체를 생성하기 위한 기본 정보(데이터, 함수)를 가지고 있는 코드이다.
- ✓ 클래스 객체를 생성하기 위한 설계코드로 얘기할 수 있다.
- ✓ 클래스를 이용하여 생성되는 객체를 다른 이름으로 인스턴스(instance)라고 한다.

9-1 클래스 기본

- 클래스 구현

- ✓ 파이썬에서 클래스를 선언하기 기본 형식은 아래와 같다.

class Person(object) :

- ✓ Class 예약어로 클래스 정의를 시작한다.
- ✓ 소괄호는 상속 받을 클래스 이름을 넣으면 된다.

- 멤버 변수

- ✓ 멤버 변수를 선언하기 위해서는 `__init__()`를 사용한다.
- ✓ `self` 변수는 클래스로 생성된 인스턴스 주소 정보이다.
- ✓ `self` 뒤의 매개변수들은 실제로 클래스가 가지게 되는 멤버 변수이다.

- 멤버 함수

- ✓ 멤버 함수는 클래스의 다양한 동작을 정의할 수 있다.

9-1 클래스 기본

● 인스턴스

- ✓ 클래스에서 인스턴스를 생성 방법은 다음과 같다. 먼저 클래스 이름을 사용하여 `__init__()` 함수의 매개변수에 맞추어 값을 입력하면 된다.

pobject = Person() :

- ✓ 인스턴스(객체)의 속성에 접근하기 위해서는 `.을` 이용한다.

```
class Person:
    def __init__(self):
        print('init_call')
        self.name = 'kim'
        self.address = 'seoul'

    def print_info(self):
        print('name:{} address:{}'.format(self.name, self.address))

pObject = Person()
print(pObject.name)
print(pObject.address)
pObject.print_info()
```

[결과]
init_call
kim
seoul
name:kim address:seoul

9-2 클래스 속성

● 클래스 속성

- ✓ 아래 코드와 같이 Student 클래스에 바로 bag 속성을 넣고, insert_bag()를 구성한다.
- ✓ 인스턴스 두 개를 생성 한 뒤 각각 인스턴스의 insert_bag()를 이용하여 특성을 알아본다.

```
class Student:
    bag = []

    def insert_bag(self, stuff):
        self.bag.append(stuff)

    def bag_info(self):
        print(self.bag)
```

```
hong = Student()
hong.insert_bag('book')
yun = Student()
yun.insert_bag('pencil')
```

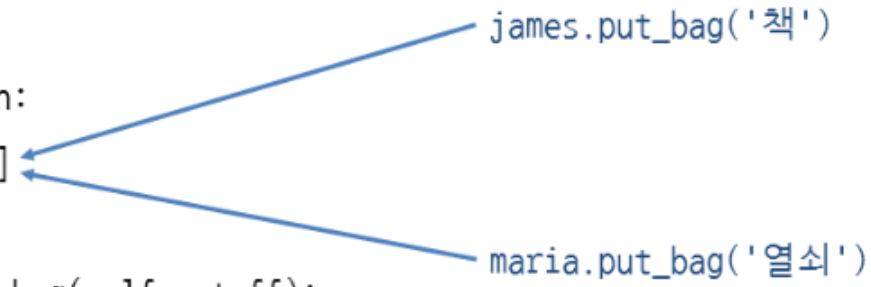
```
hong.bag_info()
hong.bag_info()
```

[결과]

```
['book', 'pencil']
['book', 'pencil']
```

```
class Person:
    bag = []

    def put_bag(self, stuff):
        self.bag.append(stuff)
```



james.put_bag('책')

maria.put_bag('열쇠')

9-2 클래스 속성

● 클래스 속성

- ✓ 클래스 속성 bag에 접근할 때 self를 사용할 수 있지만 클래스 속성을 할 때 정확한 표현은 클래스 이름을 사용하는 것이다.

```
class Student:
    bag = []

    def insert_bag(self, stuff):
        Student.bag.append(stuff)

    def bag_info(self):
        print(Student.bag)
```

```
hong = Student()
hong.insert_bag('book')
yun = Student()
yun.insert_bag('pencil')
hong.bag_info()
yun.bag_info()
```

[결과]
['book', 'pencil']
['book', 'pencil']

```
class Student:
    def __init__(self):
        self.bag = []

    def insert_bag(self, stuff):
        self.bag.append(stuff)

    def bag_info(self):
        print(self.bag)
```

```
hong = Student()
hong.insert_bag('book')
yun = Student()
yun.insert_bag('pencil')
hong.bag_info()
yun.bag_info()
```

[결과]
['book']
['pencil']

9-3 정적 메소드와 클래스 메소드

● 정적 메소드

- ✓ 정적 메서드는 사용하기 위해서는 메서드 위에 @staticmethod를 붙여야 한다.
- ✓ 정적 메서드는 선언과 동시에 메모리에 올라가 사용 가능하기 때문 매개변수에 self를 지정하지 않는다.

```
class Calc:  
    @staticmethod  
    def plus(a, b):  
        print(a + b)  
  
    @staticmethod  
    def multi(a, b):  
        print(a * b)
```

```
Calc.plus(10,20)  
Calc.multi(10,20)
```

[결과]

30

200

- ✓ 정적 메서드는 인스턴스 속성에 접근할 수 없다.
- ✓ 인스턴스를 변화시키지 않으면서 공통을 목적으로 사용하는 메서드를 만들 때 사용한다.

9-3 정적 메소드와 클래스 메소드

● 클래스 메소드

- ✓ 클래스 메서드는 사용하기 위해서는 메서드 위에 @classmethod를 붙여야 한다.
- ✓ 클래스 메서드는 첫 번째 매개변수에 cls를 지정해서 사용한다.
- ✓ cls는 클래스 속성에 접근할 때 사용한다.

```
class Car:
    count = 0

    def __init__(self):
        Car.count += 1

    @classmethod
    def maked_car_count(cls):
        print('{}대의 자동차가 생산되었습니다.'.format(cls.count))

c1 = Car()
c2 = Car()
Car.maked_car_count()
```

[결과]

2대의 자동차가 생산되었습니다.