클래스와 객체

1 객체지향 프로그래밍(Object-Oriented Programming)

① 객체지향 프로그래밍과 절차지향 프로그래밍

절차지향 프로그래밍

- 위에서 아래, 순서대로 실행
- 프로그램이 유기적으로 연결
- 대표적으로 C언어가 있음

장점	단점
• 순서대로 실행되기 때문에 실행 속도가 빠름	• 유지보수가 어려움 코드의 재사용이 어려움

클래스와 객체

1 객체지향 프로그래밍(Object-Oriented Programming)

① 객체지향 프로그래밍과 절차지향 프로그래밍

객체지향 프로그래밍

- 컴퓨터 프로그래밍의 패러다임 중 하나
- 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것
- 프로그램을 유연하고 변경이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용
- 프로그램 개발과 유지보수가 간편하고 직관적인 코드 분석이 가능
- 추상화, 캡슐화, 정보은닉, 상속, 다형성, 동적 바인딩, 오버로딩 등의 특성을 가짐

클래스와 객체

1 객체지향 프로그래밍(Object-Oriented Programming)

② 프로그래밍을 할 때 주의할 점

프로그래밍을 할 때, 코드의 복사와 붙여넣기를 자주한다면?

➡ 그만큼 코드 중복이 많이 된다는 것을 의미

1. 코드 수정 시 많은 곳을 수정해야 하고, 수정하는 과정에 실수가 발생할 수 있음
2. 중복된 코드를 줄이기 위해 함수를 사용
3. 이런 함수가 많아지게 되면, 함수 또한 의미를 파악하기 힘들어짐

1 같은 코드를 반복하지 않음

2 한번 작성한 코드는 언제든 바뀔 수 있다는 것을 생각

클래스와 객체

1 객체지향 프로그래밍(Object-Oriented Programming)

3 객체지향 프로그래밍의 이해

예 학생 관리 프로그램의 작성

학생이 한 명일 때

```
student_name = '김학생'
student_number = 2019123
student_age = 21
```

학생이 여러 명일 때

```
student_name = '김학생'
student_number = 2019123
student_age = 21

student_name2 = '이학생'
student_number2 = 2019124
student_age2 = 21

student_name3 = '최학생'
student_number3 = 2019125
student_age3 = 22

student_name4 = '박학생'
student_number4 = 2019126
student_age4 = 24

student_name5 = '홍학생'
student_number5 = 2019127
student_age5 = 22
```

클래스와 객체

1 객체지향 프로그래밍(Object-Oriented Programming)

3 객체지향 프로그래밍의 이해

예 학생 관리 프로그램의 작성

파이썬을 좀 더 활용한다면?

- 리스트와 사전 자료형을 활용

```
students =[
    {"student_name" : '김학생', "student_number" : 2019123, "student_age" : 21},
    {"student_name" : '이학생', "student_number" : 2019124, "student_age" : 21},
    {"student_name" : '최학생', "student_number" : 2019125, "student_age" : 22},
    {"student_name" : '박학생', "student_number" : 2019126, "student_age" : 24}
]
```

학생들의 평균 나이를 구하려면?

```
age_sum = 0
for student in students:
    age_sum += student['student_age']
print(int(age_sum/len(students)))
```

```
22
```

클래스와 객체

1 객체지향 프로그래밍(Object-Oriented Programming)

3 객체지향 프로그래밍의 이해

예 학생 관리 프로그램의 작성

학생이 추가된다면?, 관리 요소가 추가된다면? …

```
def create_student(name,number,age):
    return {"student_name" : name, "student_number" : number, "student_age" : age}

def sum_age(student):
    temp = 0
    for i in student:
        temp += i['student_age']
    return temp

def average_age(student):
    return sum_age(student)/len(student)
```

학생들의 성적 평균을 구하려면?
총합을 구하려면? 순위를 구하려면? …

- 관련된 함수를 점점 더 많이 만들게 되고, 자주 사용됨

조금 더 효율적인 것이 없을까? 바로 '클래스'

클래스와 객체

2 클래스와 이름공간

① 클래스의 정의

객체를 조금 더 효율적으로 생성하기 위해 만들어진 구문

- class 클래스 이름 :
  클래스 내용
- 대소문자를 구별하기 때문에 소문자로 class 정의
- 인스턴스 : 클래스로부터 만들어진 객체

```
a = 0
class S1:
    a = 1

x = S1()
print(a)
print(x.a)
```

```
0
1
```

클래스와 객체

2 클래스와 이름공간

② 클래스의 이름공간

클래스는 **별도의 이름공간**이 할당

```
class Simple_Class:
    pass

c1 = Simple_Class()
c2 = Simple_Class()

c1.a = 10
print(c1.a)
print(c2.a)
```

```
10

---------------------------
AttributeError
```

클래스와 객체

2 클래스와 이름공간

② 클래스의 이름공간

인스턴스 또한 **별도의 이름공간**을 할당

- 동적으로 인스턴스 내부에 멤버 추가 가능
- 인스턴스마다 모두 독립적인 이름공간

```
a = 0
class S1:
    a = 1

print("S1 : ", S1.a)
x = S1()
print("x.a : ", x.a)
print("---------")
x.a = 10
print("a : ", a)
print("x.a : ", x.a)
print("S1.a : ", S1.a)
```

```
S1 :  1
x.a :  1
---------
a :  0
x.a :  10
S1.a :  1
```

클래스의 메소드

1 메소드의 정의와 호출

① 메소드의 정의

클래스가 가지고 있는 함수

- 일반적인 함수와 똑같이 정의하지만 첫 번째 매개변수로 self를 사용 (관례적)
- self는 인스턴스 객체 자신의 레퍼런스를 지니고 있음

➡ 각 인스턴스들은 self를 활용해 자신의 이름공간에 접근 가능

```
class MyClass:
    def class_set(self, v):
        self.value = v

    def class_get(self):
        return self.value
```

클래스의 메소드

1 메소드의 정의와 호출

2 메소드의 호출

인스턴스 객체를 활용한 메소드 호출
(self 인자 활용)

클래스 객체를 이용한 메소드 호출

```
c = MyClass()
c.class_set('10')
print(c.class_get())

c = MyClass() # 인스턴스 생성
MyClass.class_set(c, '10')
print(MyClass.class_get(c))
```

```
10
10
```

인스턴스 객체를 활용해 메소드 호출 (self 인자는 생략)

클래스 객체를 활용해 메소드 호출 (직접 인스턴스를 적어줌)

클래스의 메소드

1 메소드의 정의와 호출

3 객체 내부 메소드의 호출

객체 내부의 메소드를 호출할 수 있음

```
class MyClass:
    def class_set(self, v):
        self.value = v

    def class_get(self):
        return self.value

    def class_incr(self):
        # 내부 메소드 호출
        self.class_set(self.value + 1)
```

```
c = MyClass()
c.class_set(1)
print(c.class_get())

print()

c.class_incr()
print(c.class_get())
```

```
1

2
```

클래스의 메소드

1 메소드의 정의와 호출

3 객체 내부 메소드의 호출

주의! self를 적어주지 않으면,
외부에서 해당 메소드를 찾게 됨

```
def class_set(i):
    print("외부 함수입니다. - ", i)

class MyClass:
    def class_set(self, v):
        self.value = v

    def class_get(self):
        return self.value

    def class_incr(self):
        # 내부 메소드 호출
        class_set(self.value + 1)
```

```
c = MyClass()
c.class_set(1)
print(c.class_get())

print()

c.class_incr()
print(c.class_get())
```

```
1

외부 함수입니다. -  2
1
```

클래스의 메소드

1 메소드의 정의와 호출

④ 정적 메소드(Static Method)

인스턴스 객체와 무관하게 클래스 이름공간에 존재하는 메소드

- 클래스 이름을 이용하여 직접 호출 가능
- 장식자 @staticmethod 사용

```
class C:
    def ham(self, x, y):
        print('instance method', x, y)

c = C()
c.ham(1, 2)

# 인스턴스 객체 없이 클래스에서 직접 호출
C.ham(1, 2)
```

```
instance method 1 2

---------------------------------------
TypeError
<ipython-input-96-de6525c4052c> in <mo
      6 c.ham(1,2)
```

```
class D:
    @staticmethod
    def spam(x, y):  # self가 없음
        print('static method', x, y)

# 인스턴스 객체 없이 클래스에서 직접 호출
D.spam(1, 2)

d = D()
# 인스턴스 객체를 통해서도 호출 가능
d.spam(1, 2)
```

```
static method 1 2
static method 1 2
```

클래스의 메소드

2 생성자와 소멸자

① 클래스 멤버와 인스턴스 멤버

클래스 멤버	클래스 이름공간에 생성, 모든 인스턴스에 공유

```
class Var:
    #클래스 변수
    c_mem = 100
    def f(self):
        self.i_mem = 200
```

클래스의 메소드

2 생성자와 소멸자

① 클래스 멤버와 인스턴스 멤버

인스턴스 멤버

인스턴스 이름공간에 생성,
인스턴스마다 **독립**

```
v1 = Var()
v2 = Var()
print(Var.c_mem)
print(v1.c_mem)
print(v2.c_mem)
```

```
100
100
100
```

```
v1.c_mem = 50
print(v1.c_mem)
print(v2.c_mem)
print(Var.c_mem)
```

```
50
100
100
```

```
v1 = Var()
v2 = Var()
v1.f()

print(v1.i_mem)
print(v2.i_mem)
```

```
200

---------------------------
AttributeError
<ipython-input-106-6324c
      4
```

클래스의 메소드

2 생성자와 소멸자

② 생성자

객체(인스턴스)가 생성될 때 자동으로 호출되는 함수

- __init__으로 정의
  (__ 의 의미는 예약된 이름, __name__, __main__ 등)
- 일반적으로 객체가 보유해야 할 변수나 자원들의 초기화를 하는 코드를 작성

```
class MyClass:
    def __init__(self):
        self.name = "Class"
        print('클래스가 생성되었습니다.', self.name)
c = MyClass()
```

```
클래스가 생성되었습니다. Class
```

클래스의 메소드

2 생성자와 소멸자

3 소멸자

객체(인스턴스)가 소멸될 때 자동으로 호출되는 함수

- __del__으로 정의
- 일반적으로 객체가 점유하고 있는 메모리나 기타 자원들의 해제를 하는 코드 작성

```
class MyClass():
    def __init__(self):
        self.name = "Class"
        print('클래스가 생성되었습니다.', self.name)

    def __del__(self):
        print('클래스가 소멸되었습니다.')

c = MyClass()
```

```
클래스가 생성되었습니다. Class
클래스가 소멸되었습니다.
```

Run! 프로그래밍

Mission | time 모듈, 클래스의 생성자와 소멸자를 활용하여 객체의 생성·소멸 시간 출력

```
from time import ctime, sleep

class Life:
    def __init__(self):
            self.birth = ctime()
            print('Birthday', self.birth)

    def __del__(self):
            print('Deathday', ctime())

def test():
    mylife = Life()
    print('Sleeping for 3 sec')
    sleep(3)

test()
```