

# Python 프로그래밍

## 클래스 II

기술경영전문대학원  
김영민

# Object oriented programming

- 사물, 대상을 뜻함 – 자동차, 사람, 냉장고, 기차, 아파트...
- 보이는 것 뿐만 아니라 개념적인 것도 포함함 – 수업, 학문, 운동, 음악...
- 실제 세계가 객체(object)들로 구성되는 것처럼 프로그램도 객체 단위로 작성하고자 함

**data before action**

- 객체 지향 프로그래밍(Object-Oriented Programming: OOP)
- Python, C++은 객체 지향 언어 (C언어는 절차 지향 언어)
- 실제 세계가 객체(object)들로 구성되는 것처럼 프로그램도 객체 단위로 작성함 – (객체 예)  
자동차, 사람, 냉장고, 기차, 아파트 ...
- C언어를 사용할 때처럼 순차적으로 죽 코딩을 하는 것이 아니라 객체를 중심으로 생각하고 객체 내부에 데이터와 기능(동작)을 포함하도록 함
- 구현하고자 하는 기능들을 객체의 입장에서 생각하여 정의함

- 내가 서점에서 책을 사는 상황을 프로그램으로 구현한다고 가정



나 : 구매자

## 데이터

소유 금액  
소유하고 있는 책들

상태

## 동작

데이터를 초기화 한다  
책을 산다

행위

소유 금액을 알려준다  
소유하고 있는 책들을 알려준다

서점



## 데이터

판매 수익  
보유하고 있는 책들

## 동작

데이터를 초기화 한다  
책을 판다

판매 수익을 알려준다  
보유하고 있는 책들을 알려준다



책

## 데이터

책 제목  
책 가격

## 동작

데이터를 초기화 한다  
책 제목을 알려준다  
책 가격을 알려준다

- 내가 테마파크 놀러가서 놀이기구 타는 상황을 구현



## 절차적 프로그램

내가 놀이기구 타는 action을 우선으로 생각한다.  
놀이기구 타는 action은 놀이기구 선택, 계산, 타는 과정으로 구성된다.

놀이기구 이름들을 배열 name에 저장하고 각 놀이기구 가격을 배열 price에 저장할 수 있다.

## 객체지향 프로그램

Action 이전에 데이터를 먼저 생각한다.

놀이기구 타는 상황을 구현하기 위해 어떤 객체가 있을 수 있는 지 생각한다.

놀이기구 타는 사람, 놀이기구, 테마파크 등이 있을 수 있다.

- 클래스는 객체를 생성하기 위한 틀, 설계도
- 객체가 포함하는 데이터와 기능을 정의함: 변수와 함수 사용
- 클래스
  - 우리가 정의하는 *데이터 타입*
  - 객체를 만들기 위한 틀
  - 데이터와 함수로 구성
- 객체(오브젝트)
  - 클래스로 선언하여 만든 실제 데이터 인스턴스(instance)

```
Class Car :  
    ... //내부에 정의
```

```
Car myCar()
```

- 사각형의 가로와 세로 길이를 받아서 넓이를 계산하는 프로그램

## 절차적 프로그램

Action: 사각형 넓이를 계산해야해~  
가로와 세로를 받아야지  
콘솔에서 인터랙티브하게 받자  
필요 변수 선언하고 (가로, 세로)  
print 로 필요한 데이터 요청, input으로 받자  
그리고 계산

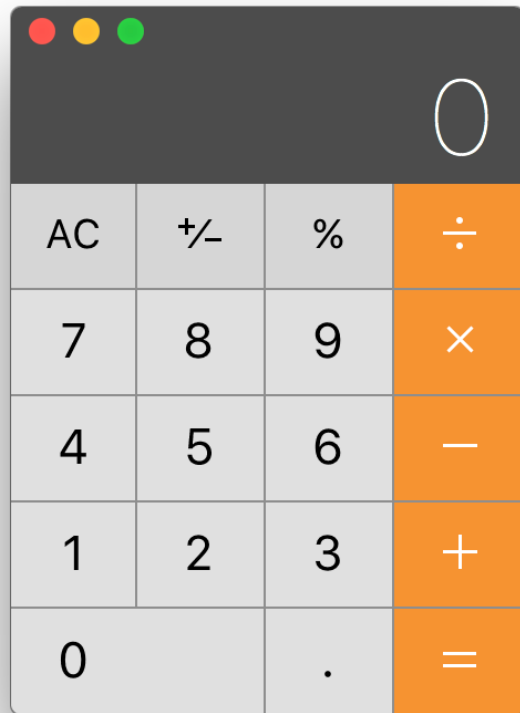
## 객체지향 프로그램

Data: 사각형 넓이를 계산하는 프로그램에서 필요한 객체가 뭐가 있을까?  
아무래도 사각형이겠지  
그럼 사각형을 디자인해보자  
사각형을 표현하는 데 필요한 데이터는?  
-> 가로,세로  
사각형이 할 수 있는 일은?  
-> 데이터 입력 받는 거  
-> 사각형이 가진 데이터 보여주는 거  
-> 넓이를 계산해주는 거  
애들로 변수와 함수를 만들어보자



# Class

- 계산기 프로그램
  - 계산을 수행하며 이전에 계산된 결과를 메모리에 저장하고 있어야 함



점프 투 파이썬 <https://wikidocs.net/book/1>

```
result = 0
```

```
def adder(num):  
    global result  
    result += num  
    return result
```

전역변수

```
print(adder(3))  
print(adder(4))
```

```
In[4]: print(adder(4))  
7  
In[5]: print(adder(5))  
12
```

# 만약 계산기가 두 개 필요하다면?

```
1  result1 = 0
2  result2 = 0
3
4  def adder1(num):
5      global result1
6      result1 += num
7      return result1
8
9  def adder2(num):
10     global result2
11     result2 += num
12     return result2
13
14  print(adder1(3))
15  print(adder1(4))
16  print(adder2(3))
17  print(adder2(7))
```

```
3
7
3
10
```

- 클래스 사용, 계산기 객체 여러 개 생성

```
1 class Calculator:
2     def __init__(self):
3         self.result = 0
4
5     def adder(self, num):
6         self.result += num
7         return self.result
8
9 cal1 = Calculator()
10 cal2 = Calculator()
11
12 print(cal1.adder(3))
13 print(cal1.adder(4))
14 print(cal2.adder(3))
15 print(cal2.adder(7))
```

- 클래스는 일종의 틀이며 그걸로 만드는 것이 객체(object)

- 클래스로 만든 객체를 인스턴스라고도 함
- 객체와 인스턴스는 용법에서 약간의 차이
  - cal1은 객체
  - cal1은 Calculator의 인스턴스
- 즉, 인스턴스는 어떤 클래스로 만들었는 지가 명시될 때 주로 사용함

```
ymkim — Python — 80x10
[>>>
[>>> class Service:
[...     secret = "지 구 는 4006년 에 멸 망 한 다 ." <-- 변수
[...
[>>> an = Service()
[>>>
[>>> an.secret
'지 구 는 4006년 에 멸 망 한 다 .'
[>>>
>>> ]
```

- 파이썬에서는 클래스 변수가 객체를 통해 접근하는 것 뿐 아닌, 클래스를 통한 직접 접근도 가능

```
ymkim — Python — 80x10
[>>>
[>>> Service.secret
'지 구 는 4006년 에 멸 망 한 다 .'
[>>>
[>>> Service.secret = '지 구 가 4006년 에 멸 망 한 다 는 사 실 은 뽕 이 다 .'
[>>>
[>>> print(an.secret)
지 구 가 4006년 에 멸 망 한 다 는 사 실 은 뽕 이 다 .
[>>>
>>> ]
```



```
ymkim — Python — 80x12
[>>> class Service:
...     secret = "지구는 4006년에 멸망한다"
...     def sum(self, a, b):
...         result = a+b
...         print("%s + %s = %s이다 ." % (a, b, result))
...
>>>
>>> an = Service()
>>>
>>> an.sum(1,1)
1 + 1 = 2이다 .
>>>
```

- 위에서 sum 함수는 3개의 매개변수를 필요로 하는 함수이지만, 사용할 때는 self를 제외한 나머지 값만 받음
- an.sum(1,1) 로 함수를 호출 시, self라는 인수에는 호출할 때 이용했던 객체 an이 자동으로 전달되기 때문

- 클래스 함수는 메서드라고도 부름
- 메서드의 첫번째 파라미터는 자신을 전달하므로 주로 self라는 이름 사용. 다른 이름을 써도 작동은 하나 가독성이 떨어짐.

- Service 클래스 업그레이드

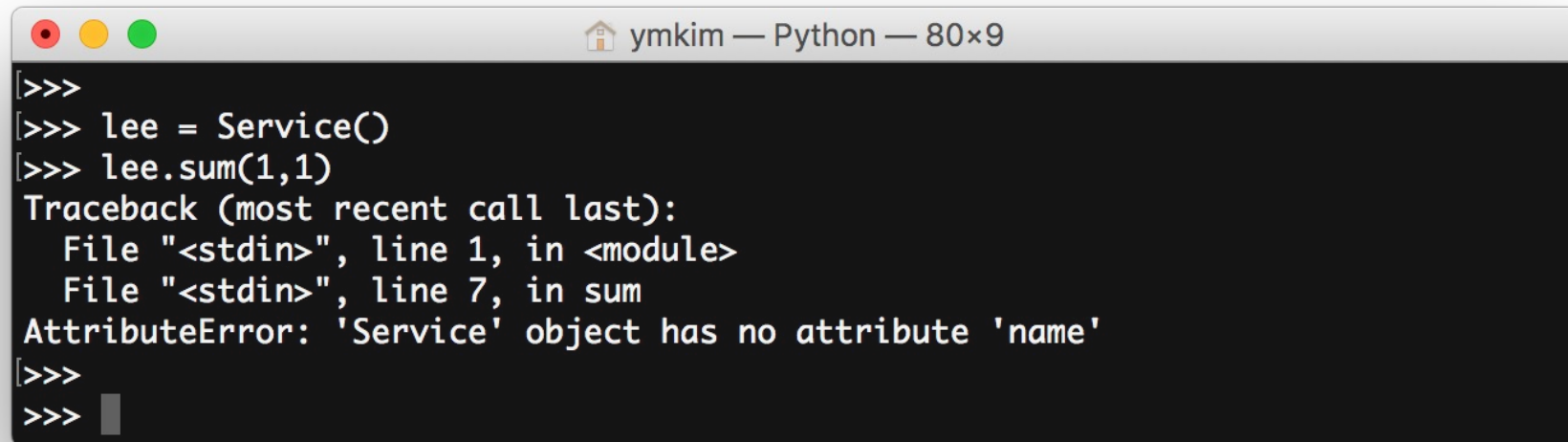
```
ymkim — Python — 80x13
>>> class Service:
...     secret = "지구는 4006년에 멸망한다"
...     def setname(self, name):
...         self.name = name
...     def sum(self, a, b):
...         result = a+b
...         print("%s님, %s + %s = %s입니다." % (self.name, a, b, result))
...
>>> an = Service()
>>> an.setname("박달도사")
>>> an.sum(1,1)
박달도사님, 1 + 1 = 2입니다.
>>>
```

- an이라는 객체에서 setname을 호출하여 an.name = name 이 실행
- 위와 같은 방법으로 객체변수를 생성할 수 있음
- 객체변수는 객체만의 변수를 의미함

- 클래스 변수는 공유되는 변수, 객체 변수는 객체별로 고유한 값 저장

```
ymkim — Python — 83x19
[>>>
[>>> kim = Service()
[>>> park = Service()
[>>> kim.name = "김 정보 "
[>>> park.name = "박 용 합 "
[>>>
[>>> print(kim.name)
김 정보
[>>> print(park.name)
박 용 합
[>>>
[>>> kim.secret = "비밀은 없다 "
[>>> print(park.secret)
지구는 4006년에 멸망한다
[>>> print(kim.secret)
비밀은 없다
[>>> Service.secret
'지구는 4006년에 멸망한다 '
[>>> ]
```

- 지금까지로는 다음과 같은 문제 발생
  - setname을 호출 하지 않는 경우



```
ymkim — Python — 80x9
[>>>
[>>> lee = Service()
[>>> lee.sum(1,1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in sum
AttributeError: 'Service' object has no attribute 'name'
[>>>
>>> ]
```

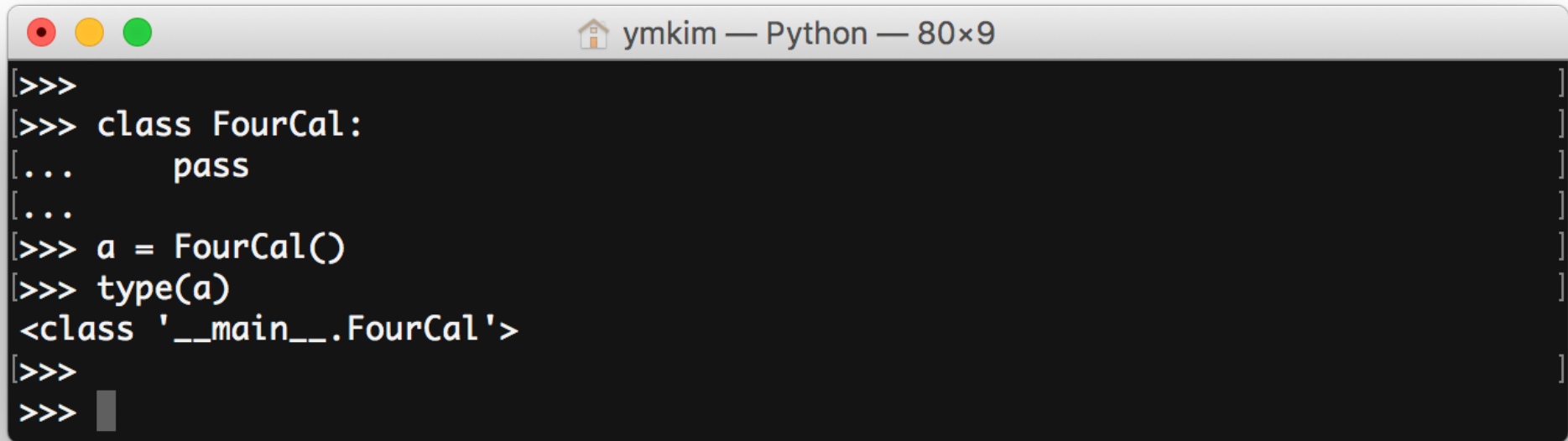
- 객체를 생성하면서 동시에 이름을 입력하도록

```
ymkim — Python — 80x9
>>>
>>> class Service:
...     secret = "지구는 4006년에 멸망한다"
...     def __init__(self, name):
...         self.name = name
...     def sum(self, a, b):
...         result = a+b
...         print("%s님, %s + %s = %s입니다." % (self.name, a, b, result))
... 
```

```
ymkim — Python — 80x5
>>>
>>> an = Service("박달도사")
>>> an.sum(1, 1)
박달도사님, 1 + 1 = 2입니다.
>>> 
```

```
class 클래스이름[(상속 클래스명)]:  
    <클래스 변수 1>  
    <클래스 변수 2>  
    ...  
    def 메서드1(self[, 인수1, 인수2,,,]):  
        <수행할 문장 1>  
        <수행할 문장 2>  
        ...  
    def 메서드2(self[, 인수1, 인수2,,,]):  
        <수행할 문장1>  
        <수행할 문장2>  
        ...  
    ...
```

- 사칙연산을 할 수 있는 클래스 FourCal
- 우선 아무것도 안하는 껍데기 클래스를 만들어보자.



```
ymkim — Python — 80x9
>>>
>>> class FourCal:
...     pass
...
>>> a = FourCal()
>>> type(a)
<class '__main__.FourCal'>
>>>
>>>
```



```
ymkim — Python — 80x14
>>>
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...
>>> a = FourCal()
>>> a.setdata(4,2)
>>>
>>> print(a.first)
4
>>> print(a.second)
2
>>> 
```

```
ymkim — Python — 80x14
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...     def sum(self):
...         result = self.first + self.second
...         return result
...
>>> a = FourCal()
>>> a.setdata(4,2)
>>>
>>> print(a.sum())
6
>>>
```

```
class FourCal:
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def sum(self):
        result = self.first + self.second
        return result
    def mul(self):
        result = self.first * self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result
    def div(self):
        result = self.first / self.second
        return result
```

```
In[10]: a = FourCal()
In[11]: b = FourCal()
In[12]: a.setdata(4,2)
In[13]: b.setdata(3,7)
In[14]: a.sum()
Out[14]: 6
In[15]: a.mul()
Out[15]: 8
In[16]: a.sub()
Out[16]: 2
In[17]: a.div()
Out[17]: 2.0
In[18]: b.sub()
Out[18]: -4
```

# Class example

- 두 명의 플레이어가 세 개의 주사위를 굴리고, 많은 점수가 나오는 사람이 이긴다.



```
import random

player1_dice = []
player2_dice = []

for i in range(3):
    player1_dice.append(random.randint(1, 6))
    player2_dice.append(random.randint(1, 6))

print("Player 1 rolled" + str(player1_dice))
print("Player 2 rolled" + str(player2_dice))

if sum(player1_dice) == sum(player2_dice):
    print("Draw")
elif sum(player1_dice) > sum(player2_dice):
    print("Player 1 wins")
else:
    print("Player 2 wins")
```

<https://www.raspberrypi.org/magpi/class-python/>

```
from random import randint

class Player:
    def __init__(self):
        self.dice = []

    def roll(self):
        self.dice = [] # clears current dice
        for i in range(3):
            self.dice.append(randint(1, 6))

    def get_dice(self):
        return self.dice

player1 = Player()
player2 = Player()

player1.roll()
player2.roll()

print("Player 1 rolled" + str(player1.get_dice()))
print("Player 2 rolled" + str(player2.get_dice()))

if sum(player1.get_dice()) == sum(player2.get_dice()):
    print("Draw!")
elif sum(player1.get_dice()) > sum(player2.get_dice()):
    print("Player 1 wins!")
else:
    print("Player 2 wins!")
```

<https://www.raspberrypi.org/magpi/class-python/>

```
class Customer(object):
    """A customer of ABC Bank with a checking account. Customers have the
    following properties:

    Attributes:
        name: A string representing the customer's name.
        balance: A float tracking the current balance of the customer's account.
    """

    def __init__(self, name):
        """Return a Customer object whose name is *name*."""
        self.name = name

    def set_balance(self, balance=0.0):
        """Set the customer's starting balance."""
        self.balance = balance

    def withdraw(self, amount):
        """Return the balance remaining after withdrawing *amount*
        dollars."""
        if amount > self.balance:
            raise RuntimeError('Amount greater than available balance.')
        self.balance -= amount
        return self.balance

    def deposit(self, amount):
        """Return the balance remaining after depositing *amount*
        dollars."""
        self.balance += amount
        return self.balance
```



<https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/>