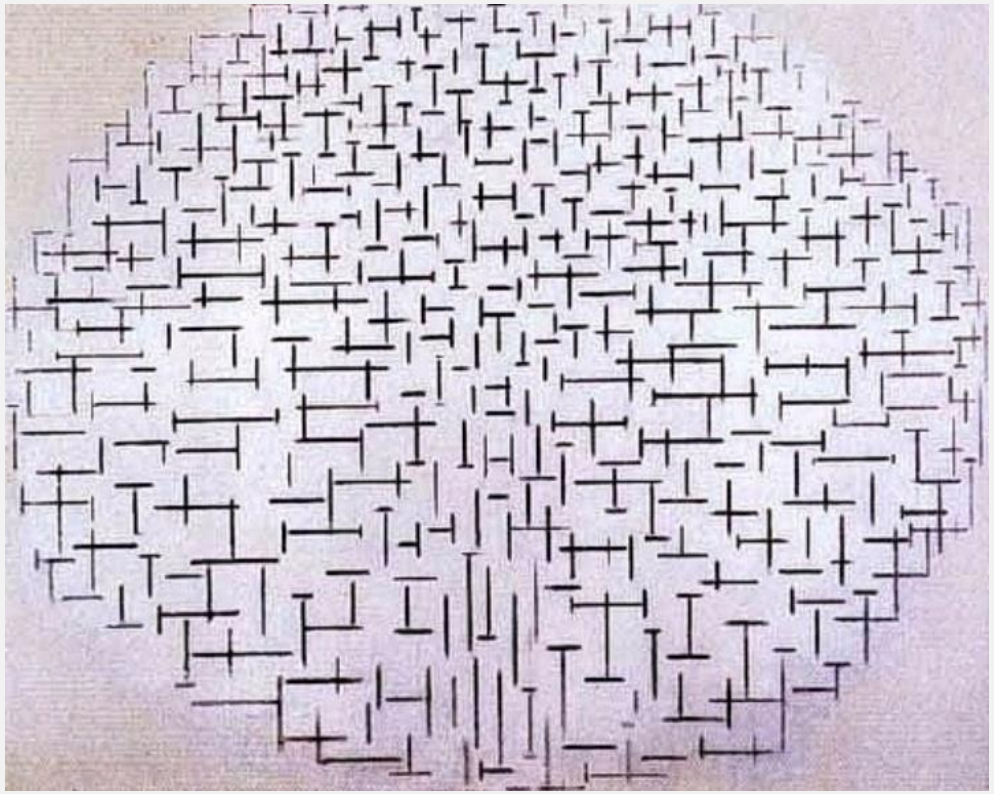
# Class와 Object

By 박수현

# Programming Paradigm

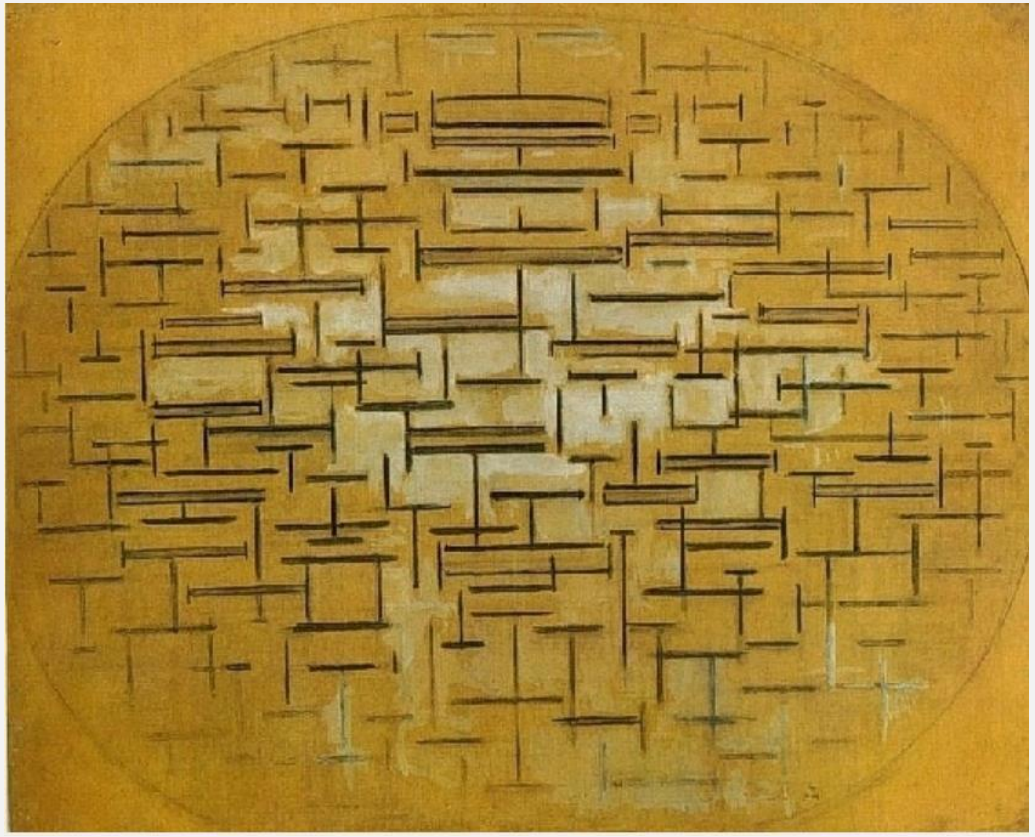
- 다음 그림은 무엇을  그린 것일까 ?



http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search

# Programming Paradigm

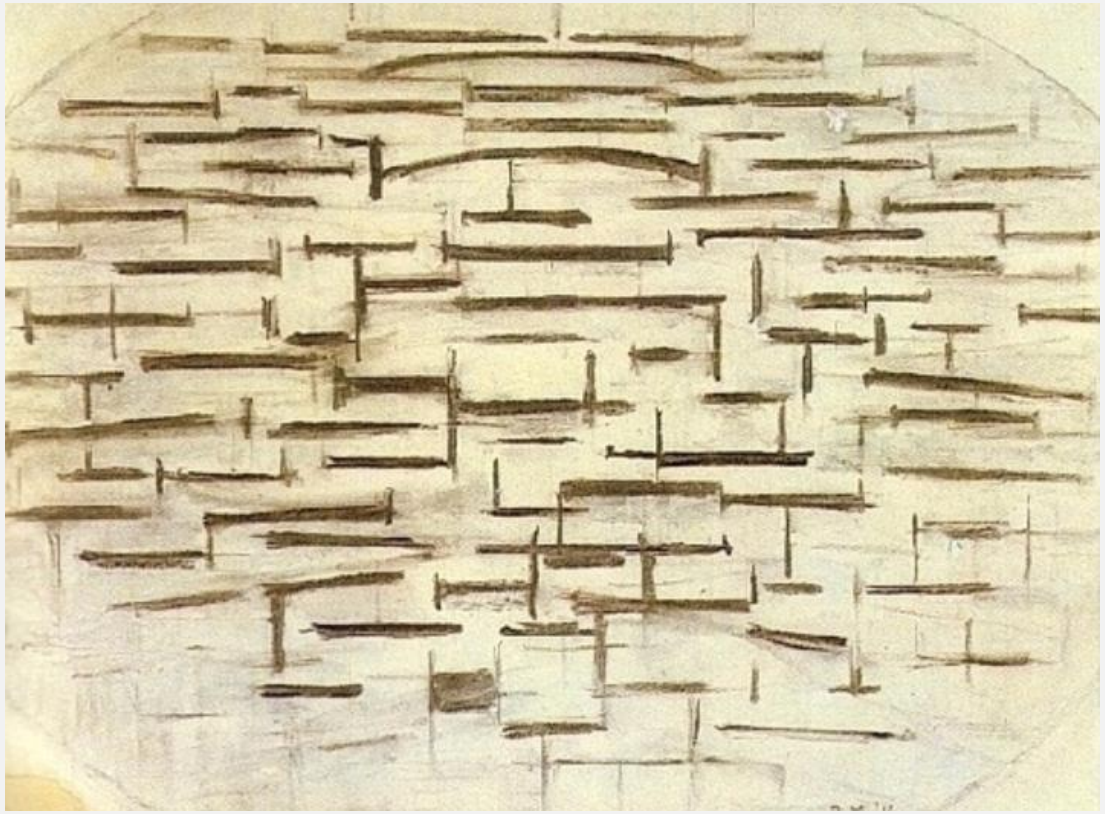
- 다음 그림은 무엇을  그린 것일까 ?



http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search

# Programming Paradigm

- 다음 그림은 무엇을  그린 것일까 ?



http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search

# Programming Paradigm

- 다음 그림은 무엇을  그린 것일까 ?

http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search

# Programming Paradigm

- 다음 그림은 무엇을  그린 것일까 ?

http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search

# Programming Paradigm

- 다음 그림은 무엇을  그린 것일까 ?

http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search

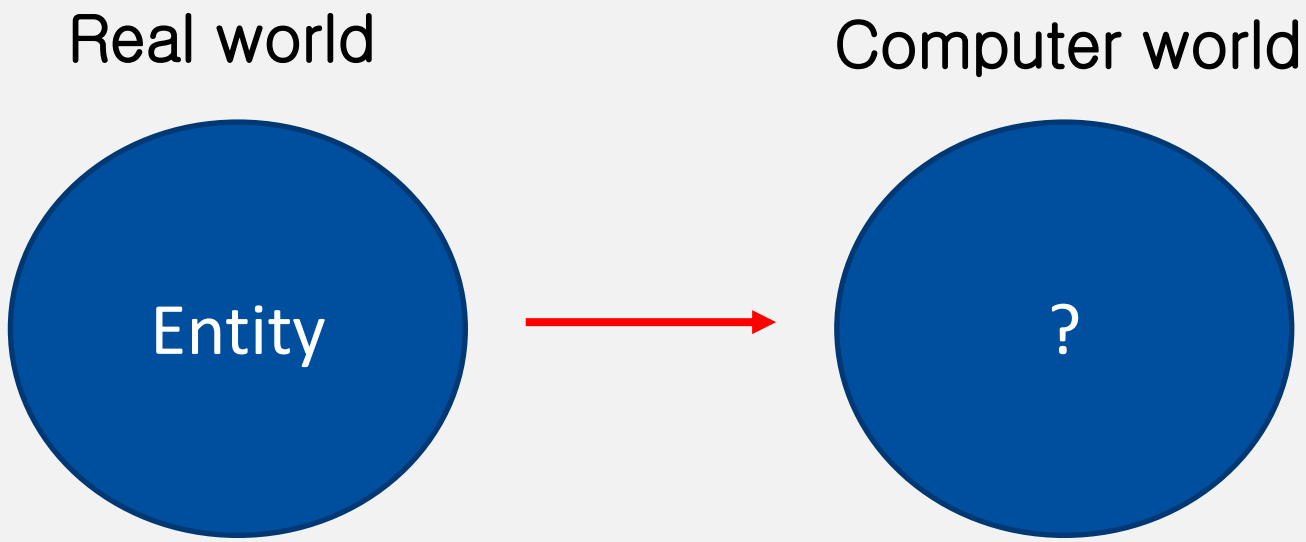
# Programming Paradigm

- 추상화 (抽象畵) - abstract painting

- 추상화 (抽象化) – abstraction

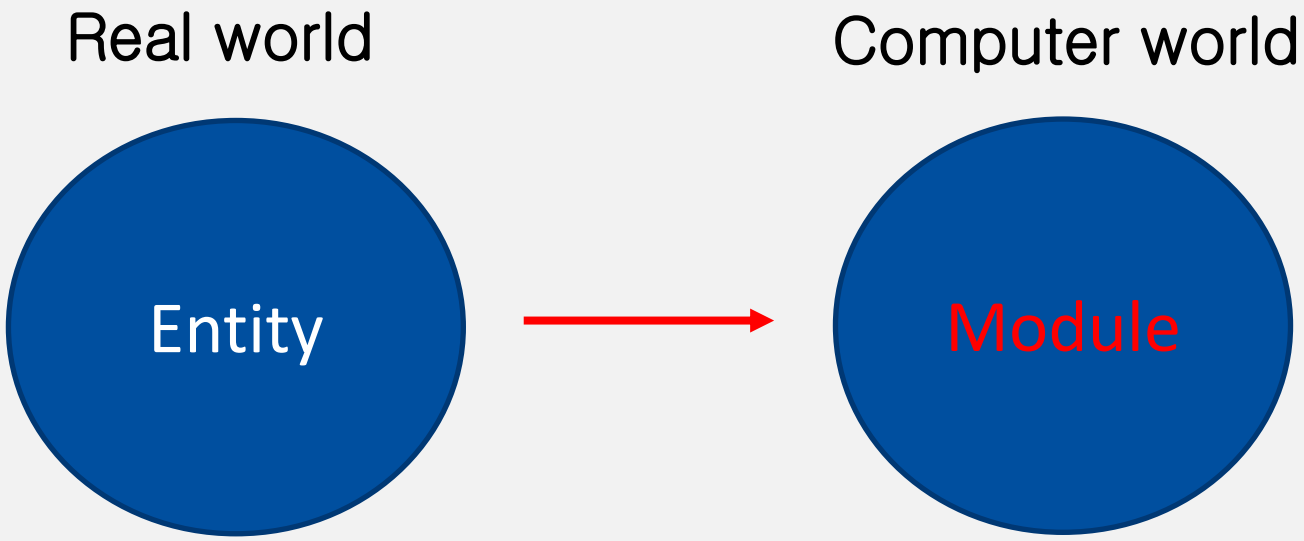
# Programming Paradigm

- Mapping / Modeling / 추상화 (抽象化, abstraction)



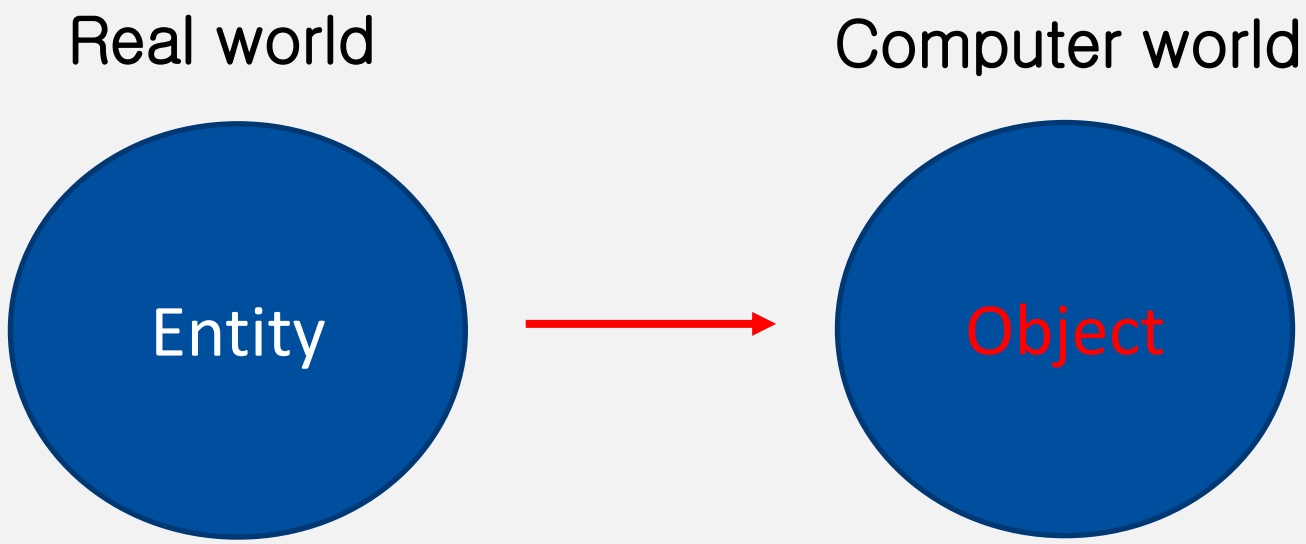
# Programming Paradigm

- **Structure Programming (SP)**
  - Module
  - Structure 기반
  - Readability (가독성 : 可讀性)



# Programming Paradigm

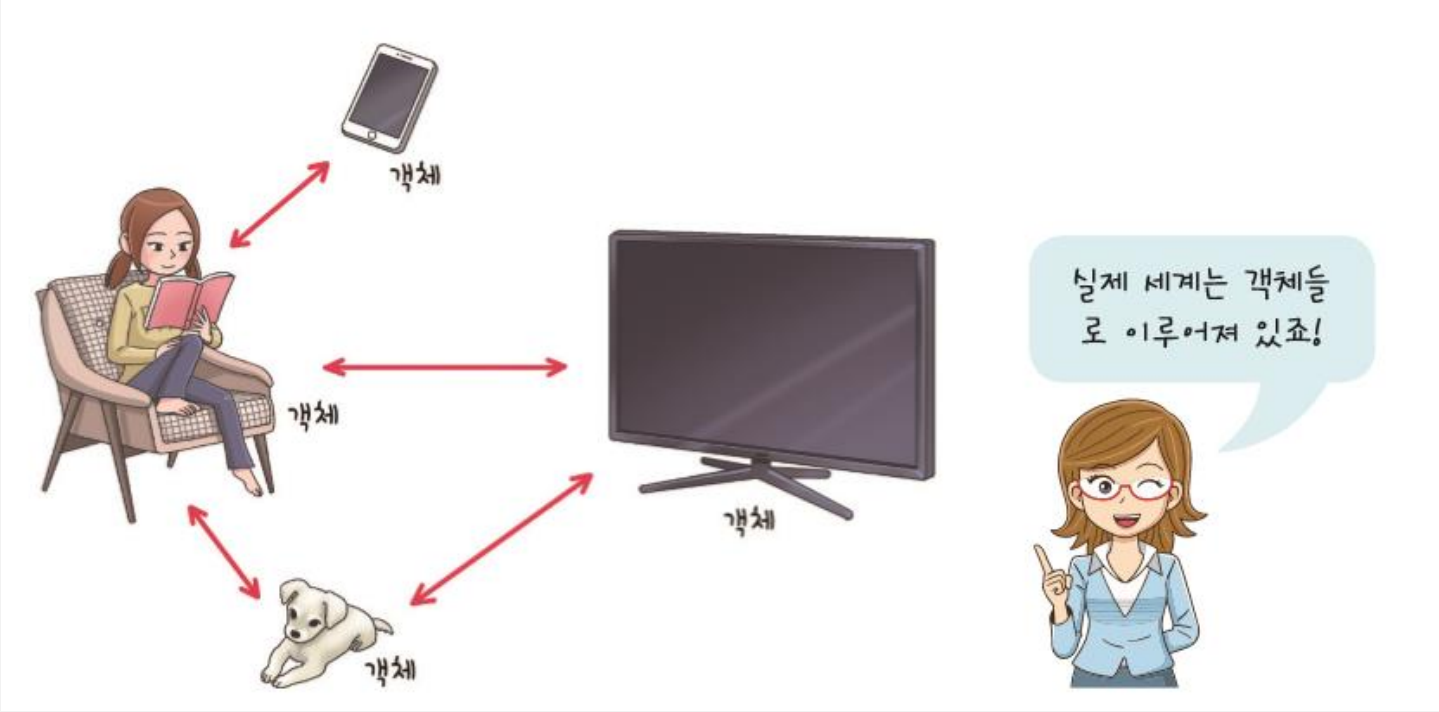
- **Object-Oriented Programming (OOP)**
  - Object
  - Encapsulation (from information hiding)
  - Reusability (재사용성 : 再使用性)



# 객체지향 프로그래밍

- **Object-Oriented Programming (OOP)**

  Software를 객체(object)로 구성하는 programming paradigm



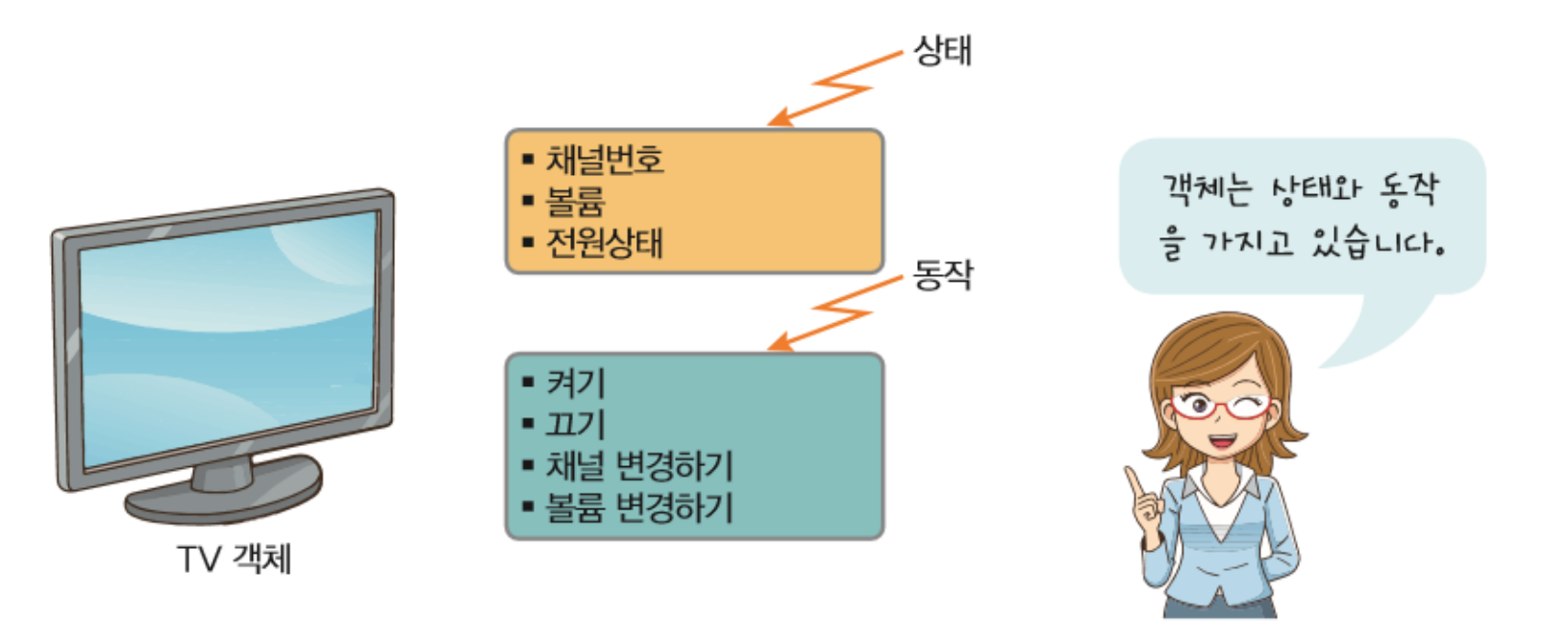
# 객체 (Object)

- 객체(Object) : 개체(entity)를 구성(정의)하는 속성(attribute, data)과 그 속성을 읽거나 변경(read / write)할 수 있는 operation (function)을 모아놓은 것.
  - Object = id + attribute + method
  - Object의 다른 이름 : 인스턴스(instance)

- Attribute (member variable)
  - 개체(entity)에서 다루고자 하는 Data (or field)
- Method (member function)
  - 속성을 읽거나 변경(read / write)할 수 있는 operations (functions)
  - Data(attribute)를 manipulation할 수 있는 operations (functions)

# 객체 (Object)

- 객체는 상태와 동작을 가지고 있음
  - 객체의 상태(state)는 객체의 속성(attribute)
  - 객체의 동작(behavior)은 객체가 취할 수 있는 동작(기능 : method)



# 인스턴스 변수(Instance Variable)와 메소드(Method)

Object =
id + attribute + method

ID : Television

Attribute

- channel
- volume
- on

Methods

- turnOn()
- turnOff()
- changeChannel()
- changeVolume()

*television_simple_class.py

File Edit Format Run Options Window Help

```
class Television:  # class id

    # attributes
    channel = 0
    volume = 0
    on = "on"

    #methods
    def changeChannel(self, ch_num):
        self.channel = ch_num

    def changeVolume(self, volume_value):
        self.volume = volume_value

    def turnOn(self, on_off):
        self.on = on_off

    def display(self):
        print("₩n>> channel :", self.channel)
        print("     volume :", self.volume)
        print("     on :", self.on)

class Main:
    def main(self):
        tv = Television()

        input_channel = int(input("₩n>> Channel number : "))
        input_Volume = int(input(">> Volume : "))
        input_onoff = input(">> on or off : ")

        tv.changeChannel(input_channel)
        tv.changeVolume(input_Volume)
        tv.turnOn(input_onoff)

        tv.display()

#main
m = Main()
m.main()
```

```
>> Channel number : 18
>> Volume : 20
>> on or off : on

>> channel : 18
     volume : 20
     on : on
>>>
```

Object =
id + attribute + method

ID : Television

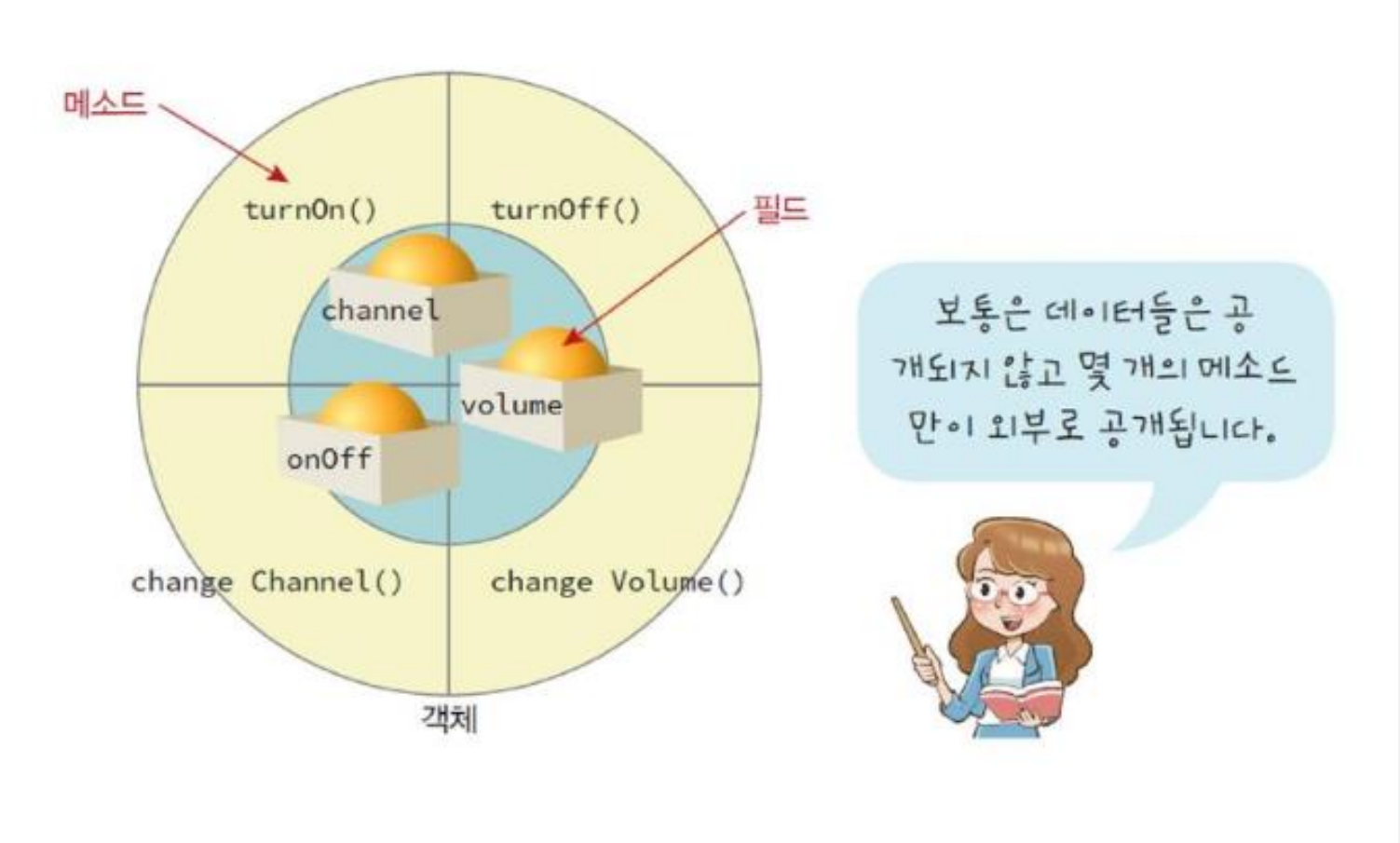
Attribute

- channel
- volume
- on

Methods

- turnOn()
- turnOff()
- changeChannel()
- changeVolume()

# 인스턴스 변수(Instance Variable)와 메소드(Method)




https://fehoon.tistory.com/101

# 객체지향 프로그래밍

- **Object-Oriented Programming (OOP) Paradigm**
  - Real world의 entity를 computer world의 object으로 mapping(modeling, abstraction)
  - Basic philosophy of OOP : maximize the level of software reusability
    - Instantiation
      - Class(template) → objects (instances)
      - instantiation
    - Inheritance
      - Polymorphism
      - Override / overload
      - Multiple inheritance
      - Interface (?)

# Class

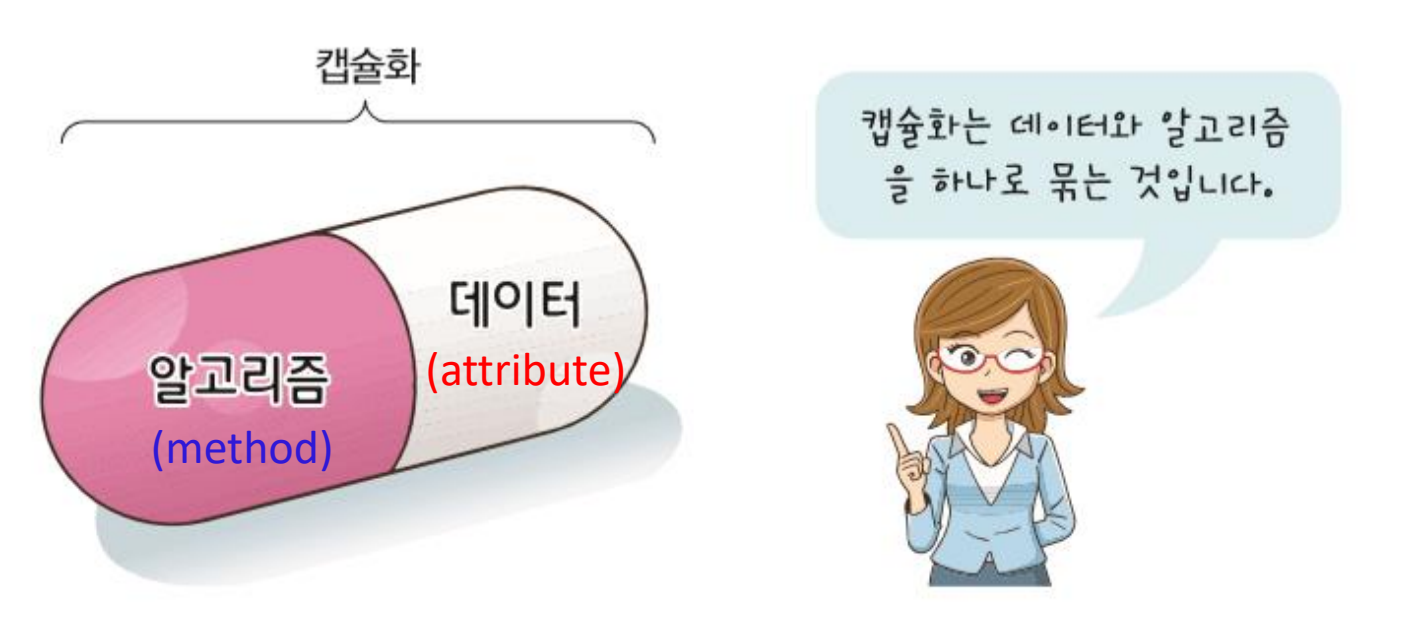
- 객체(object)에 대한 Template를 Class라고 함
- Class로부터 만들어지는 각각의 객체들을 그 Class의 인스턴스(instances)라고 함

# Class

- 객체 지향 프로그래밍(OOP)에서 특정 객체를 생성하기 위해 변수와 메소드를 정의하는 일종의 틀
  - Attribute(data) 저장하기 위한 변수들과 객체의 행동을 수행하기 위한 method들을 같은 이름공간으로 묶은 집합체

# Class

- Encapsulation(캡슐화)
  - 처리하고자 하는 data(attribute)와 이 data를 processing하는 algorithm(function, operation, method)을 하나로 묶어 공용 interfaces만 제공하고 구현 세부 사항을 감추는 것(information hiding)을 캡슐화(encapsulation)라고 함
  - Reusability(재사용성) 지원



# Class

- 점(.) 표기법
  - 객체의 이름과 속성 사이, 이름과 method 사이에 점(.)을 이용
  - Python 표기법으로 점 표기법(dot notation)이라고 함

# Class

- Class는 크게 Class id, member variable, member method 등 3 가지 부분으로 구성

```
class Ball:

    size = 0
    direction = “default”

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

Class id

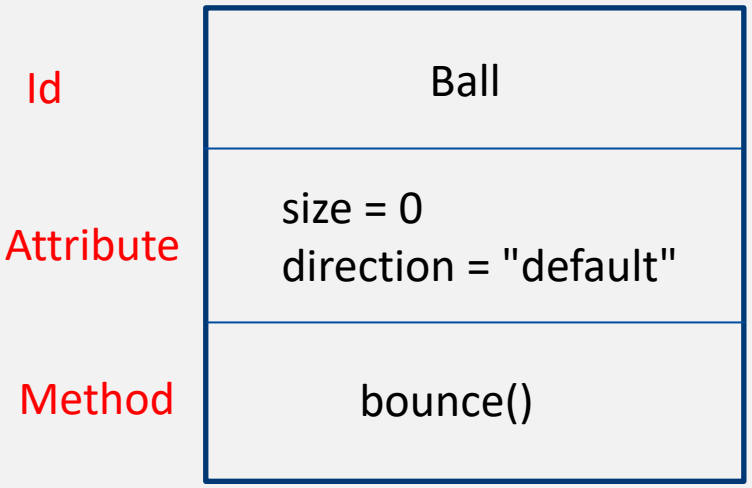
Member Variable

Member method

self는 Python 만의 독특한 변수로 class 내에서 정의되는 함수(method)는 무조건 첫번째 인자로 self를 사용하여야 함 → 후에 설명

# Class

- 간단한 Ball class




Class notation in UML

Ball.py

File Edit Format Run Options Window Help

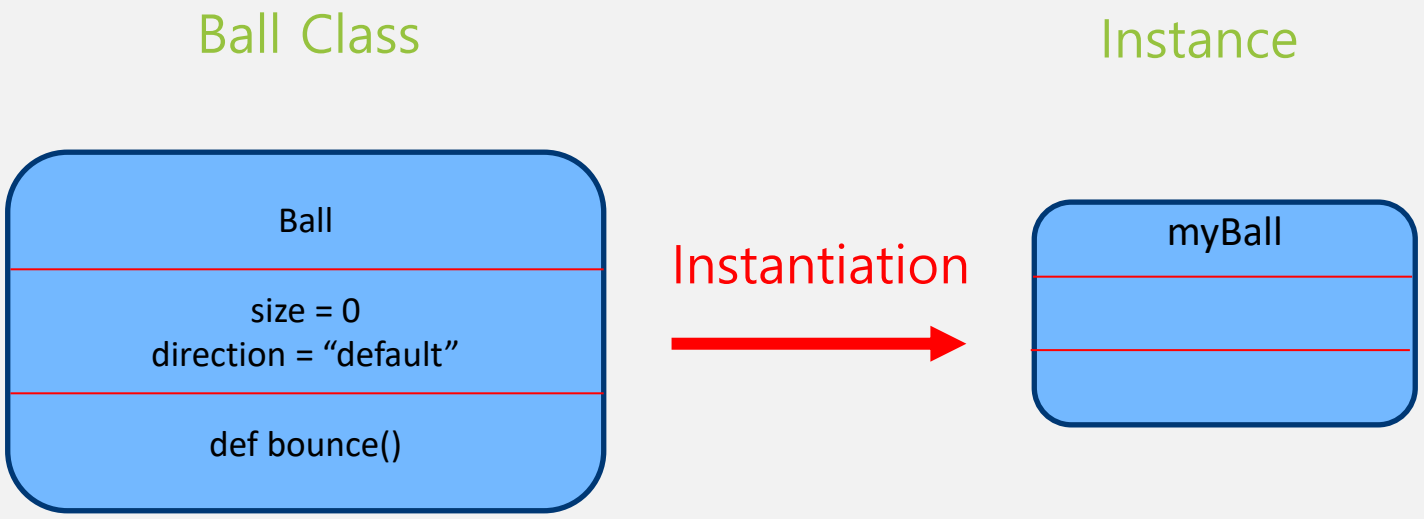
```
1  # create a simple Ball class
2
3  class Ball:                          # Class ID
4
5      size = 0                         # Member variable
6      direction = "default"
7
8      def bounce(self):                # Member function (method)
9          if self.direction == "down":
10             self.direction = "up"
11
```

  - UML : Unified Modeling Language

# Instance (Object)

- Class를 사용하여 만든 실제 객체
- Class는 instance를 만드는 하나의 틀(template)
- Instance를 통해 변수나 함수의 이름을 찾는 순서
  - Instance 영역 -> Class 영역 -> Global 영역

# Instance(Object) 생성




myBall = Ball()

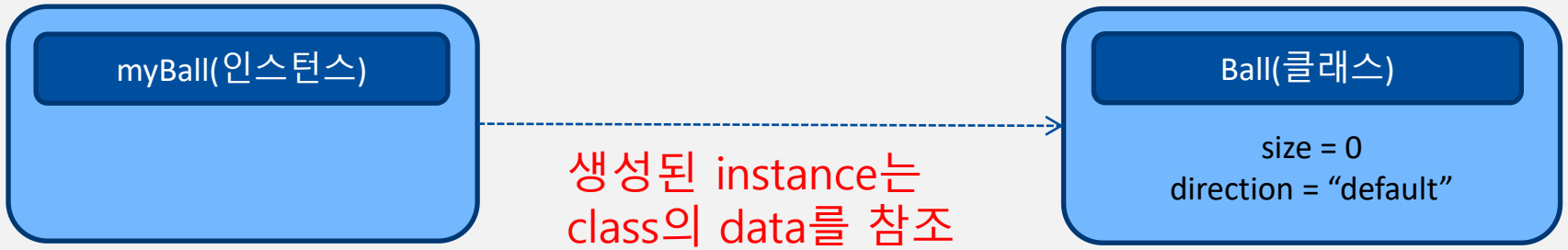
# Instance 생성

- 기본적으로 instance는 생성이 완료된 직후 원본 class와 동일한 데이터(attribute)와 함수(method)를 가짐 (참조).

```
class Ball:
    size = 0
    direction = “default”

    def bounce(self):
        if self.direction == "down":
            self.direction = "up“

myBall = Ball()
```

간단한 Ball class 생성

Class의 instance인 myBall생성



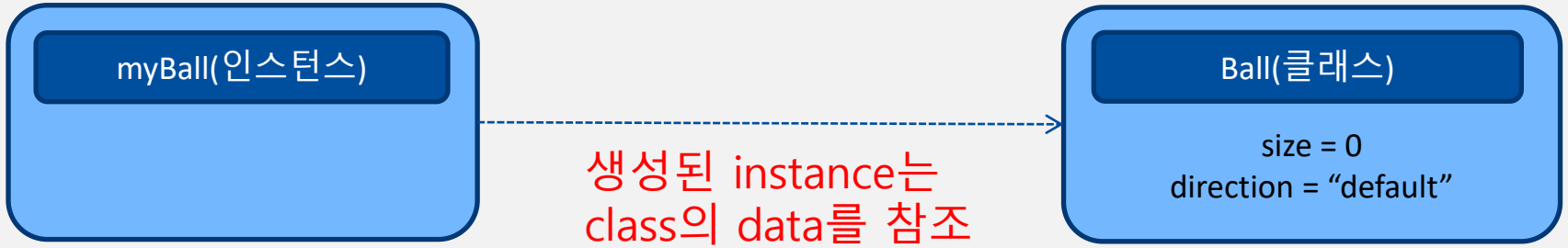
# Instance 생성

Ball-2.py

File Edit Format Run Options Window Help

```
1  # create a simple Ball class
2
3  class Ball:
4
5      size = 0
6      direction = "default"
7
8      def bounce(self):                # This is a method
9          if self.direction == "down":
10             self.direction = "up"
11
12 # main
13 myBall = Ball()
14
15 print("1) myBall.size = ", myBall.size)
16 print("2) myBall.direction = ", myBall.direction)
17
```

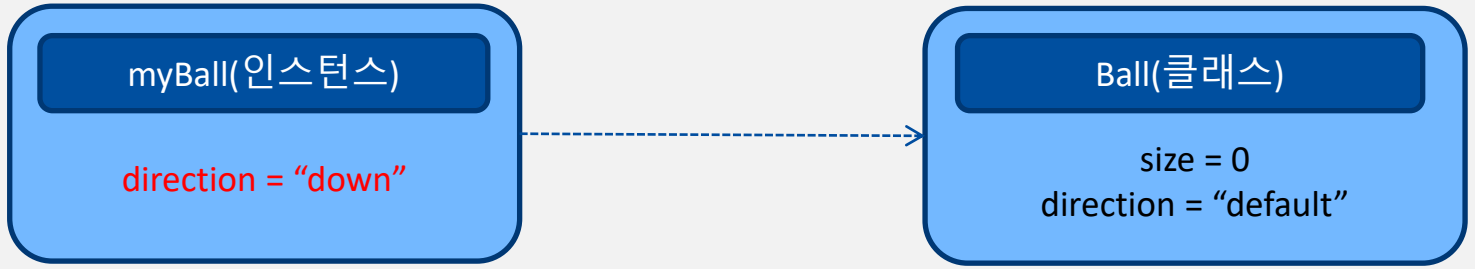
```
1) myBall.size =  0
2) myBall.direction =  default
```



# Instance attribute 값 변경

```
class Ball:
    size = 0
    direction = “default”

    def bounce(self):
        if self.direction == "down":
            self.direction = "up“

myBall = Ball()
myBall.direction = "down“
```

Instance의 멤버 변수 값 변경




Instance에 특화된 data는 instance의 이름공간(name space)에 저장

# Instance attribute 값 변경

Ball-2-1.py

File Edit Format Run Options Window Help

```
1  # create a simple Ball class
2
3  class Ball:
4
5      size = 0
6      direction = "default"
7
8      def bounce(self):                # This is a method
9          if self.direction == "down":
10             self.direction = "up"
11
12 # main
13 myBall = Ball()
14 myBall.direction = "down"
15
16 print("1) myBall.size = ", myBall.size)
17 print("2) myBall.direction = ", myBall.direction)
18
```

```
1) myBall.size =  0
2) myBall.direction =  down
```

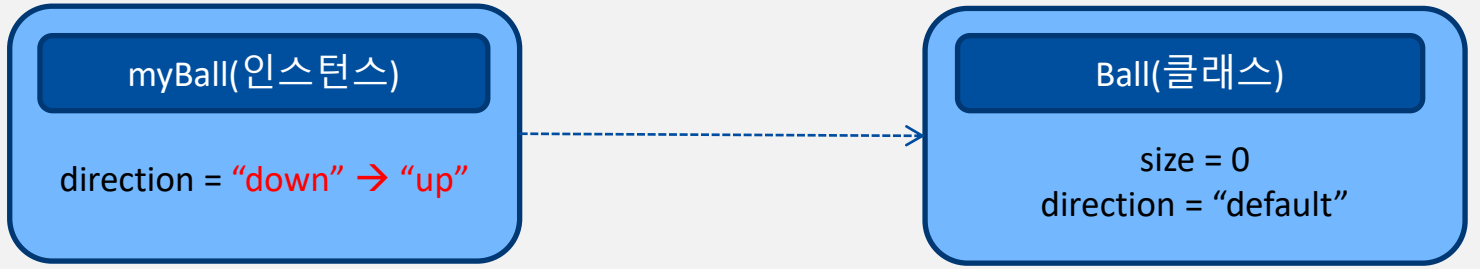

Instance에 특화된 data는 instance의 이름공간(name space)에 저장

# Instance attribute 변경 및 method 호출

```
class Ball:
    size = 0
    direction = “default”

    def bounce(self):
        if self.direction == "down":
            self.direction = "up“

myBall = Ball()
myBall.direction = "down“
myBall.bounce()
```

Instance의 멤버 변수 값 변경

Method 호출



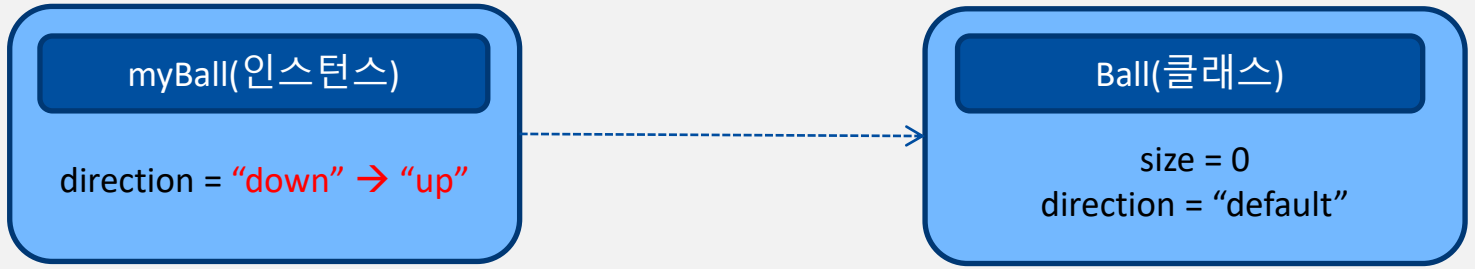

Instance에 특화된 data는 instance의 이름공간에 저장

Ball-2-2.py

File Edit Format Run Options Window Help

```
# create a simple Ball class

class Ball:

    size = 0
    direction = "default"

    def bounce(self):                # This is a method
        if self.direction == "down":
            self.direction = "up"

# main
myBall = Ball()
myBall.direction = "down"

print("1) myBall.size = ", myBall.size)
print("2) myBall.direction = ", myBall.direction)

myBall.bounce()
print("₩n3) myBall.size = ", myBall.size)
print("4) myBall.direction = ", myBall.direction)
```

```
1) myBall.size =  0
2) myBall.direction =  down

3) myBall.size =  0
4) myBall.direction =  up
```




Instance에 특화된 data는 instance의 이름공간에 저장

# Instance 멤버 변수 동적 추가

- Class와 Instance에 동적으로 member variable 추가/삭제 가능.

```
class Ball:
    size = 0
    direction = “default”

    def bounce(self):
        if self.direction == "down":
            self.direction = "up“

myBall = Ball()
myBall.direction = "down“
myBall.color = “red”
```

myBall 객체에만 color 멤버 변수를 추가

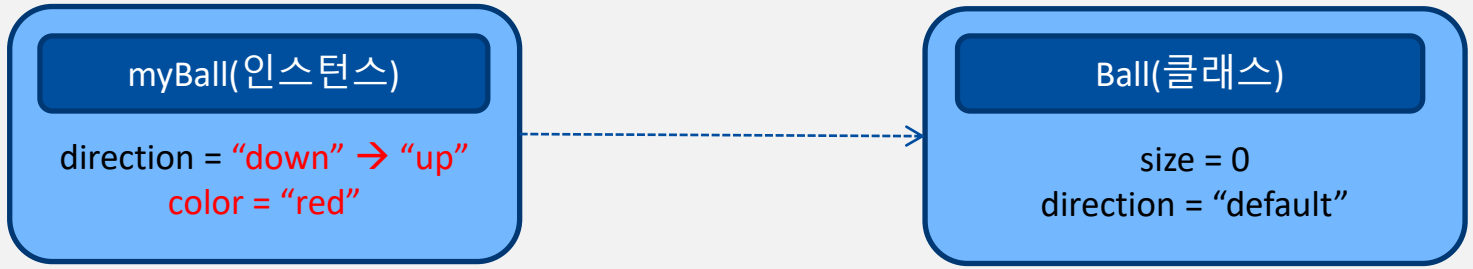

Instance에 동적으로 추가된 변수 “color”

Ball-2-3.py

File Edit Format Run Options Window Help

```
1  # create a simple Ball class
2
3  class Ball:
4
5      size = 0
6      direction = "default"
7
8      def bounce(self):                # This is a method
9          if self.direction == "down":
10             self.direction = "up"
11
12 # main
13 myBall = Ball()
14 myBall.direction = "down"
15 myBall.color = "red"
16
17 print("1) myBall.size = ", myBall.size)
18 print("2) myBall.direction = ", myBall.direction)
19 print("3) myBall.color = ", myBall.color)
20
21 myBall.bounce()
22 print("₩n4) myBall.size = ", myBall.size)
23 print("5) myBall.direction = ", myBall.direction)
24
```

```
1) myBall.size =  0
2) myBall.direction =  down
3) myBall.color =  red

4) myBall.size =  0
5) myBall.direction =  up
```




Instance에 동적으로 추가된 변수 “color”

# Instance를 통한 Class 참조

- 인스턴스가 자신을 생성한 Class를 참조하기 위해 instance의 내장 속성 '__class__'를 사용 (Built-In Class Attributes in Python)

```
class Ball:
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"

myBall = Ball()
myBall.__class__.direction = "side"
```

'__class__' 속성을 이용해 class 데이터를 변경

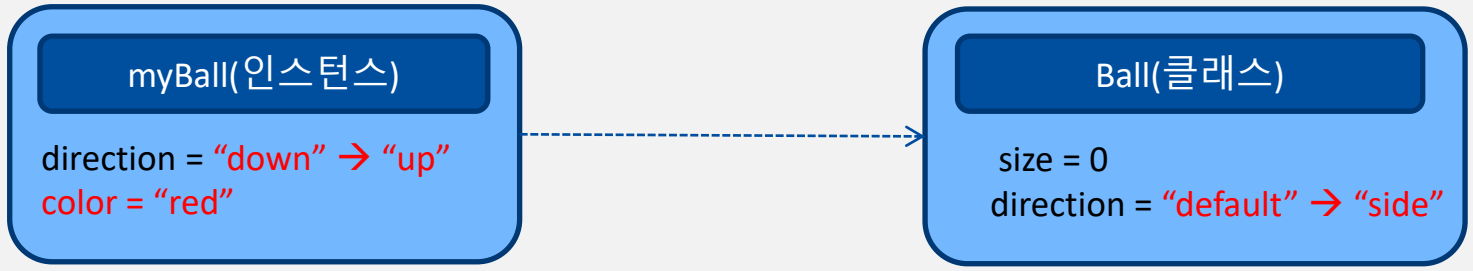

생성된 instance를 통해 class 변수 값 변경

Ball-2-4.py

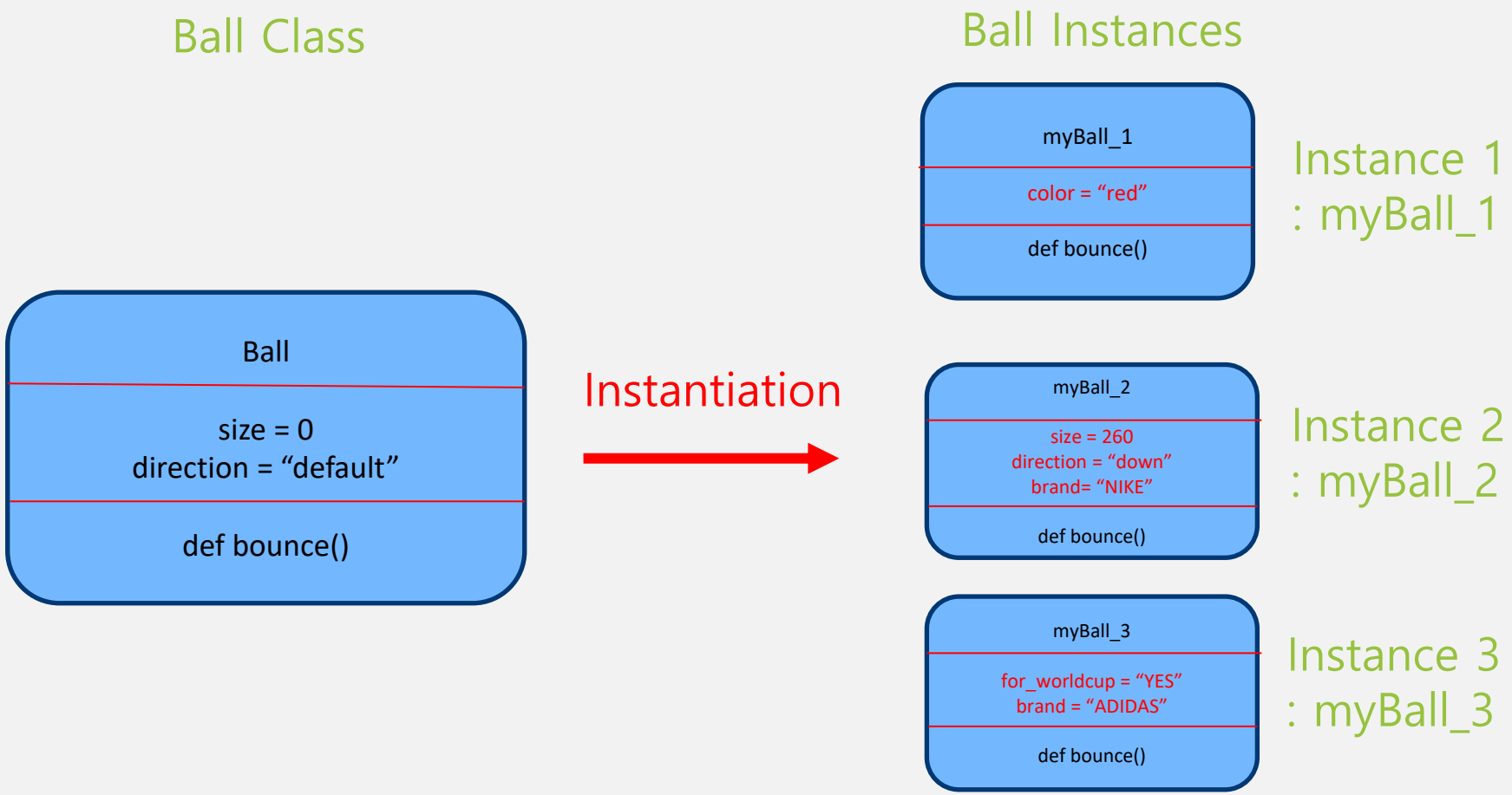
File Edit Format Run Options Window Help

```
# create a simple Ball class

class Ball:

    size = 0
    direction = "default"

    def bounce(self):               # This is a method
        if self.direction == "down":
            self.direction = "up"

# main
myBall = Ball()
myBall.direction = "down"
myBall.color = "red"

print("1) myBall.size = ", myBall.size)
print("2) myBall.direction = ", myBall.direction)
print("3) myBall.color = ", myBall.color)

myBall.bounce()
print("₩n4) myBall.size = ", myBall.size)
print("5) myBall.direction = ", myBall.direction)

print("6) Class Ball의 direction 값 = ", myBall.__class__.direction)

myBall.__class__.direction = "side"
print("7) Class Ball의 direction 값 = ", myBall.__class__.direction)

```

```
1) myBall.size =  0
2) myBall.direction =  down
3) myBall.color =  red

4) myBall.size =  0
5) myBall.direction =  up
6) Class Ball의 direction 값 =  default
7) Class Ball의 direction 값 =  side
```



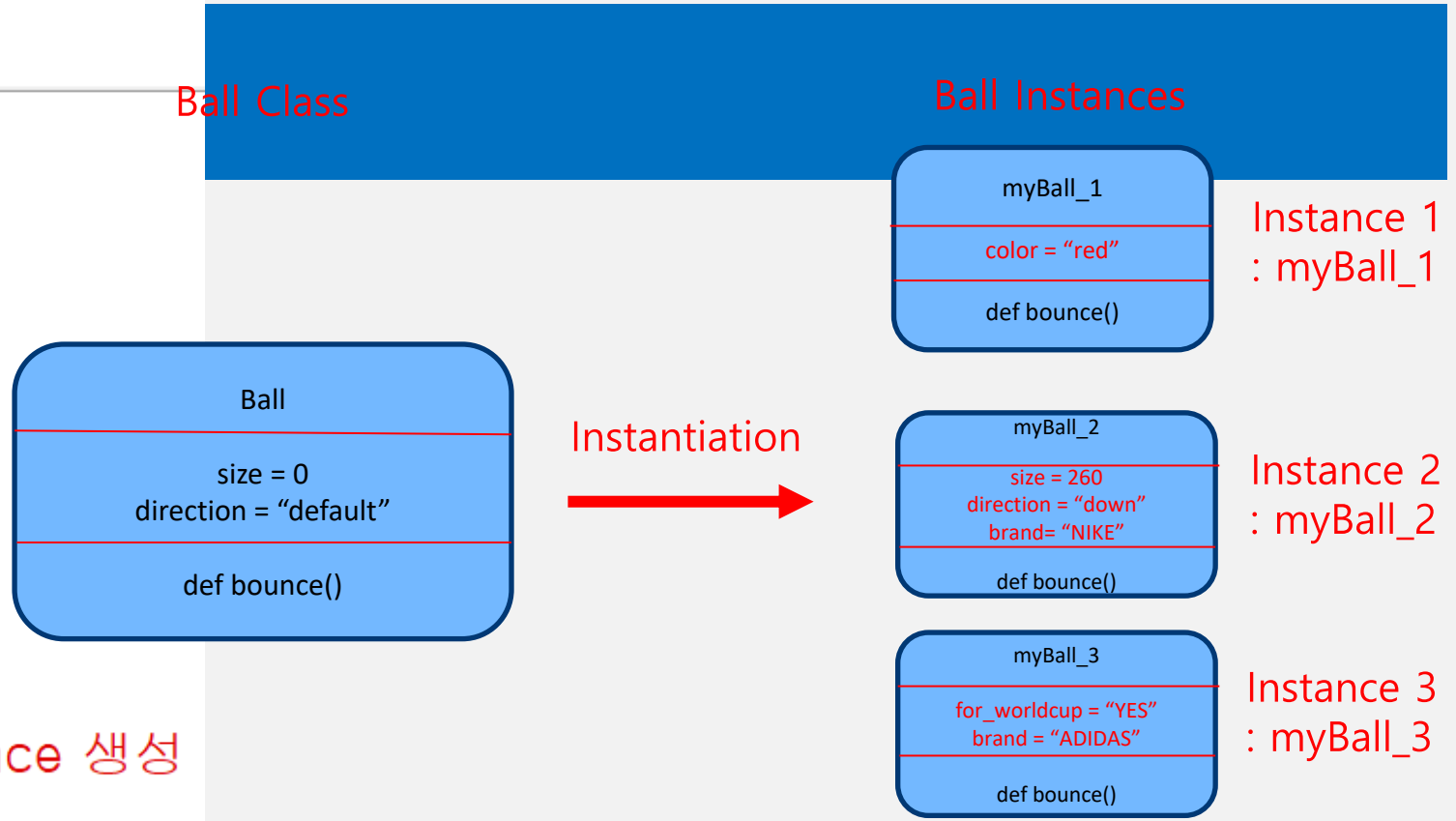
# Instance 생성


Ball Class
Ball Instances
Ball
size = 0
direction = “default”
def bounce()
Instantiation
myBall_1
color = “red”
def bounce()
Instance 1
: myBall_1
myBall_2
size = 260
direction = “down”
brand= “NIKE”
def bounce()
Instance 2
: myBall_2
myBall_3
for_worldcup = “YES”
brand = “ADIDAS”
def bounce()
Instance 3
: myBall_3

Ball-3-1.py

File Edit Format Run Options Window Help

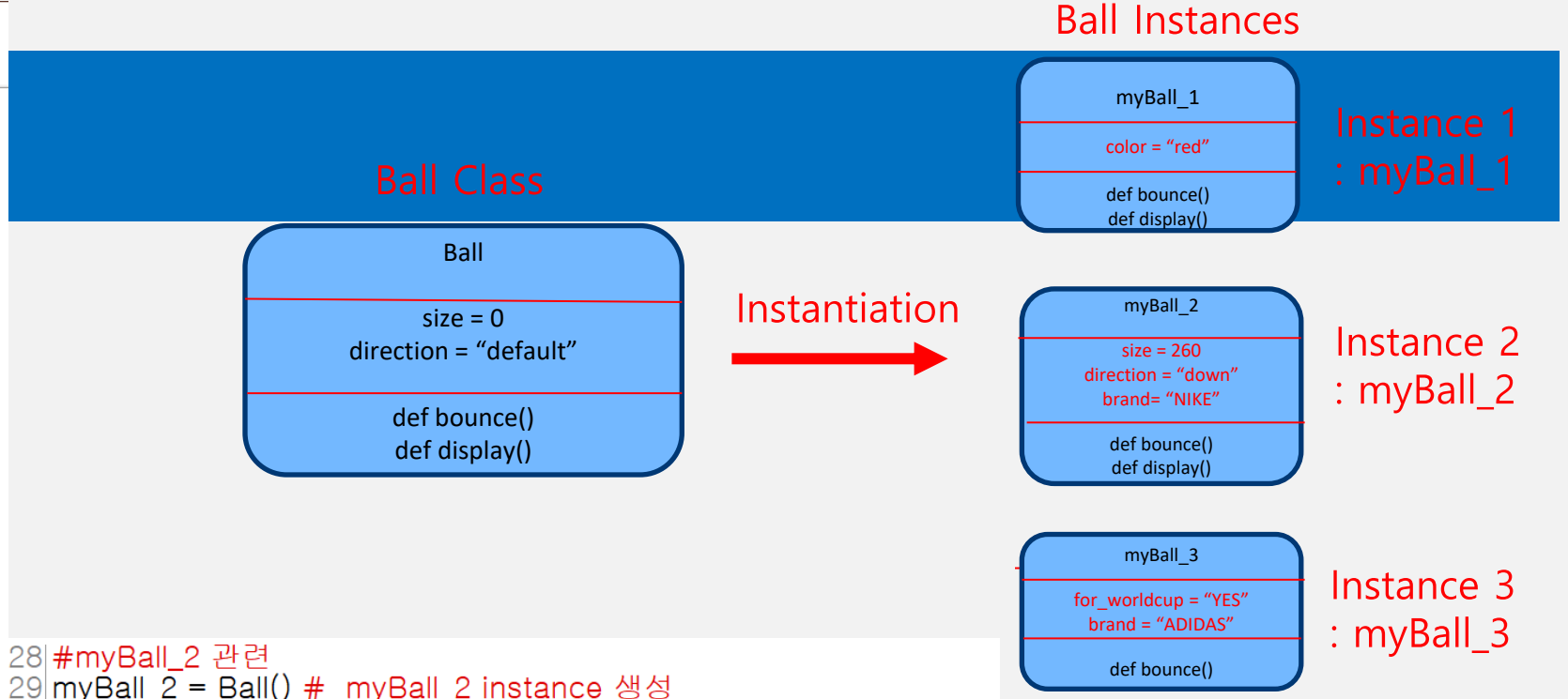
```
# using the Ball class

class Ball:

    size = 0
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"

#main
myBall_1 = Ball() #  myBall_1 instance 생성
myBall_1.color = "red"

myBall_2 = Ball() #  myBall_2 instance 생성
myBall_2.size = 260
myBall_2.direction = "down"
myBall_2.brand = "NIKE"

myBall_3 = Ball() #  myBall_3 instance 생성
myBall_3.for_worldcup = "YES"
myBall_3.brand = "ADIDAS"
```



Ball-3-2.py

File Edit Format Run Options Window Help

```python
D = True

class Ball:

    size = 0
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"

    def display(self):
        print("d1) self =", self)
        print("d2) self.size =", self.size)
        print("d3) self.direction =", self.direction)

#main
#myBall_1 관련
myBall_1 = Ball() #  myBall_1 instance 생성
if D:
    print("m1) myBall_1 =", myBall_1)

myBall_1.color = "red"
myBall_1.display()
if D:
    print("m1-1) myBall_1.color =", myBall_1.color)
```




```python
#myBall_2 관련
myBall_2 = Ball() #  myBall_2 instance 생성
if D:
    print("₩nm2) myBall_2 =", myBall_2)

myBall_2.size = 260
myBall_2.direction = "down"
myBall_2.brand = "NIKE"
myBall_2.display()
if D:
    print("m2-1) myBall_2.brand =", myBall_2.brand)

#myBall_3 관련
myBall_3 = Ball() #  myBall_3 instance 생성
if D:
    print("₩nm3) myBall_3 =", myBall_3)

myBall_3.for_worldcup = "YES"
myBall_3.brand = "ADIDAS"
myBall_3.display()
if D:
    print("m3-1) myBall_3.for_worldcup =", myBall_3.for_worldcup)
    print("m3-2) myBall_3.brand =", myBall_3.brand)
```

```
m1) myBall_1 = <__main__.Ball object at 0x000001DC303D8910>
d1) self = <__main__.Ball object at 0x000001DC303D8910>
d2) self.size = 0
d3) self.direction = default
m1-1) myBall_1.color = red

m2) myBall_2 = <__main__.Ball object at 0x000001DC30466FA0>
d1) self = <__main__.Ball object at 0x000001DC30466FA0>
d2) self.size = 260
d3) self.direction = down
m2-1) myBall_2.brand = NIKE

m3) myBall_3 = <__main__.Ball object at 0x000001DC30466BE0>
d1) self = <__main__.Ball object at 0x000001DC30466BE0>
d2) self.size = 0
d3) self.direction = default
m3-1) myBall_3.for_worldcup = YES
m3-2) myBall_3.brand = ADIDAS
```

Ball-3-3.py

File Edit Format Run Options Window Help

```python
D = True

class Ball:

    size = 0
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"

    def display(self):
        print("d1) self =", self)
        print("d2) self.size =", self.size)
        print("d3) self.direction =", self.direction)

```

```
m1) myBall_1 = <__main__.Ball object at 0x000001D993CF69D0>
d1) self = <__main__.Ball object at 0x000001D993CF69D0>
d2) self.size = 0
d3) self.direction = default
m1-1) myBall_1.color = red

m2) myBall_2 = <__main__.Ball object at 0x000001D993CF6BB0>
d1) self = <__main__.Ball object at 0x000001D993CF6BB0>
d2) self.size = 260
d3) self.direction = down
m2-1) myBall_2.brand = NIKE

m3) myBall_3 = <__main__.Ball object at 0x000001D993CF6B20>
d1) self = <__main__.Ball object at 0x000001D993CF6B20>
d2) self.size = 0
d3) self.direction = default
m3-1) myBall_3.for_worldcup = YES
m3-2) myBall_3.brand = ADIDAS
```

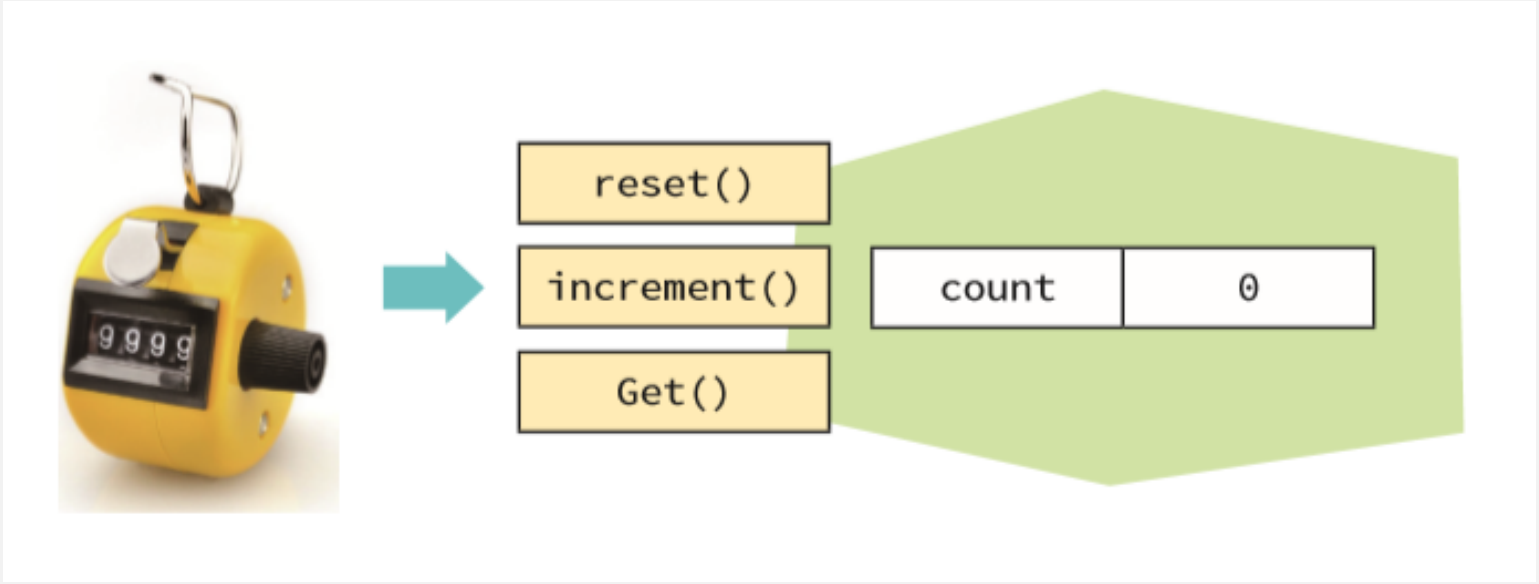
```python

class Main: # driver class

    def main(self):
        #myBall_1 관련
        myBall_1 = Ball() #  myBall_1 instance 생성
        if D:
            print("m1) myBall_1 =", myBall_1)

        myBall_1.color = "red"
        myBall_1.display()
        if D:
            print("m1-1) myBall_1.color =", myBall_1.color)

        #myBall_2 관련
        myBall_2 = Ball() #  myBall_2 instance 생성
        if D:
            print("₩nm2) myBall_2 =", myBall_2)

        myBall_2.size = 260
        myBall_2.direction = "down"
        myBall_2.brand = "NIKE"
        myBall_2.display()
        if D:
            print("m2-1) myBall_2.brand =", myBall_2.brand)

        #myBall_3 관련
        myBall_3 = Ball() #  myBall_3 instance 생성
        if D:
            print("₩nm3) myBall_3 =", myBall_3)

        myBall_3.for_worldcup = "YES"
        myBall_3.brand = "ADIDAS"
        myBall_3.display()
        if D:
            print("m3-1) myBall_3.for_worldcup =", myBall_3.for_worldcup)
            print("m3-2) myBall_3.brand =", myBall_3.brand)

#main
m = Main()
m.main()
```

# Class 형식

```
Class Class-ID:
    attribute list

    def method_1(self, argument list):
     ....

    def method_2(self, argument list):
     ....
```

# Class의 예

- Counter Class를 작성하여 보자. Counter Class는 기계식 계수기를 나타내며 경기장이나 콘서트에 입장하는 관객 수를 세기 위하여 사용할 수 있음.
- 버튼을 누를 때마다 입장하는 관객수가 1명씩 증가함.
- 사용완료후 0으로 리셋하는 기능이 있음.
- 현재의 관객수를 확인할 수 있는 기능이 필요함.



counter.py

File Edit Format Run Options Window Help

```
class Counter:

    count = 0

    def reset(self):
        self.count = 0

    def increment(self):
        self.count += 1

    def get(self):
        return self.count

def counter_operate(counter_obj, how_many_click):
    counter_obj.reset()
    print("₩nc1) counter_obj의 count 초기값 :", counter_obj.get())

    for i in range(1, how_many_click+1):
        counter_obj.increment()
        print(">> {}번째 click 후, counter_obj의 값 : {}".format(i, counter_obj.get()))

    counter_obj.reset()
    print("₩nc2) reset 버튼 click 후 counter_obj 값 :", counter_obj.get())


# main
counter1 = Counter()
counter2 = Counter()

n = int(input(">> counter1 click 횟수를 입력하시오 : "))

print("1) counter1을 {}번 click".format(n))
counter_operate(counter1, n)

m = int(input("₩n>> counter2 click 횟수를 입력하시오 : "))

print("2) counter2을 {}번 click".format(m))
counter_operate(counter2, m)

```

```
>> counter1 click 횟수를 입력하시오 : 4
1) counter1을 4번 click

c1) counter_obj의 count 초기값 : 0
>> 1번째 click 후, counter_obj의 값 : 1
>> 2번째 click 후, counter_obj의 값 : 2
>> 3번째 click 후, counter_obj의 값 : 3
>> 4번째 click 후, counter_obj의 값 : 4

c2) reset 버튼 click 후 counter_obj 값 : 0

>> counter2 click 횟수를 입력하시오 : 3
2) counter2을 3번 click

c1) counter_obj의 count 초기값 : 0
>> 1번째 click 후, counter_obj의 값 : 1
>> 2번째 click 후, counter_obj의 값 : 2
>> 3번째 click 후, counter_obj의 값 : 3

c2) reset 버튼 click 후 counter_obj 값 : 0
```

# 객체 초기화 - 생성자(constructor) method

- Class로 부터 객체 생성(instantiation)시 자동으로 실행되는 특수 method
- 객체(또는 instance)가 생성(instantiation)될 때 자동으로 호출
- 특수 method '__init__()'으로 정의.
  - 특수 method : Class 생성 시 Python에서 (자동으로) 만들어주는 method (user가 overload 할 수 있음) – p.49

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction
```

__init__() method 정의

# 객체 초기화 - 생성자(constructor) method

- 객체를 생성하면서 속성값을 설정.
- 하지만 이 기능 만이 생성자의 기능이 아님.
- Constructor argument

생성자를 통해 instance 생성 시 초기화 할 멤버변수 값을 argument로 전달

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

myBall = Ball("red", "small", "down")
```

myBall instance 생성과 동시에 myBall 멤버 변수 color="red", size="small", direction = "down" 으로 초기화

*Constructor.py

File Edit Format Run Options Window Help

```
1  class Ball:
2      def __init__(self, color, size, direction):  # Constructor(생성자)
3          print ("₩nc1) 생성자")                    # __init__() method
4          self.color = color
5          self.size = size
6          self.direction = direction
7
8          print ("c2) self.color = ",color)
9          print ("c3) self.size = ",size)
10         print ("c4) self.direction = ",direction)
11
12     def bounce(self):
13         print ("₩nb1) in bounce()")
14         if self.direction == "down":
15             self.direction = "up"
16
17 #main
18 print ("1) main")
19 myBall = Ball("red", "small", "down")   # Create an instance
20                                         # with 3 attributes
21 print ("₩n2) myBall.size :", myBall.size)
22 print ("3) myBall.color :", myBall.color)
23 print ("4) myBall.direction :", myBall.direction)
24
25 myBall.bounce()
26 print ("₩n5) myBall.direction : ", myBall.direction)
27
```

```
1) main

c1) 생성자
c2) self.color =  red
c3) self.size =  small
c4) self.direction =  down

2) myBall.size : small
3) myBall.color : red
4) myBall.direction : down

b1) in bounce()

5) myBall.direction :  up
>>>
```

# 생성자의 예

전체적인 구조

```
class 클래스 이름:
    def __init__(self, ...):
        ...
```

__init__() 메소드가 생성자이다.
여기서 객체의 초기화를 담당한다.

```
class Counter:
    def __init__(self) :
        self.count = 0
    def reset(self) :
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

counter_with_Constructor.py

File Edit Format Run Options Window Help

```python
class Counter:

    def __init__(self) :
        self.count = 0
        print("ini) self = {}. self.count = {}".format(self, self.count))

    def reset(self):
        self.count = 0
        print("res) self = {}. self.count = {}".format(self, self.count))


    def increment(self):
        self.count += 1
        print("inc) self = {}. self.count = {}".format(self, self.count))

    def get(self):
        print("get) self = {}. self.count = {}".format(self, self.count))
        return self.count

#main
print("1) Create counter1.")
counter1 = Counter()
print(">> counter1 = ", counter1)

print("₩n2) Create counter2.")
counter2 = Counter()
print(">> counter2 = ", counter2,"₩n")

counter1.reset()
print("₩n3) counter1.count =", counter1.get())

counter1.increment()
print("₩n4) counter1.count =", counter1.get())

counter2.reset()
print("₩n5) counter2.count =", counter2.get())

counter2.increment()
counter2.increment()
print("₩n6) counter2.count =", counter2.get())
```

```
1) Create counter1.
ini) self = <__main__.Counter object at 0x0000026056A969D0>. self.count = 0
>> counter1 =  <__main__.Counter object at 0x0000026056A969D0>

2) Create counter2.
ini) self = <__main__.Counter object at 0x0000026056A96FD0>. self.count = 0
>> counter2 =  <__main__.Counter object at 0x0000026056A96FD0>

res) self = <__main__.Counter object at 0x0000026056A969D0>. self.count = 0
get) self = <__main__.Counter object at 0x0000026056A969D0>. self.count = 0

3) counter1.count = 0
inc) self = <__main__.Counter object at 0x0000026056A969D0>. self.count = 1
get) self = <__main__.Counter object at 0x0000026056A969D0>. self.count = 1

4) counter1.count = 1
res) self = <__main__.Counter object at 0x0000026056A96FD0>. self.count = 0
get) self = <__main__.Counter object at 0x0000026056A96FD0>. self.count = 0

5) counter2.count = 0
inc) self = <__main__.Counter object at 0x0000026056A96FD0>. self.count = 1
inc) self = <__main__.Counter object at 0x0000026056A96FD0>. self.count = 2
get) self = <__main__.Counter object at 0x0000026056A96FD0>. self.count = 2

6) counter2.count = 2
```

# 객체 초기화 - 생성자 method (constructor method)

- Default constructor (묵시적 생성자)
  - 생성자를 두지않고도 class를 정의 할 수가 있는데, 이를 묵시적 생성자라고 함
  - 생성자를 지정해주지 않았을 때, 묵시적으로 생성되는 기본 생성자
  - 필요에 따라 이를 override / overload 하여 사용
    - 이미 앞의 예제들에서 생성자를 overload하여 사용하였음

```
def __init__(self):
    pass
```

# 소멸자 메소드 (destructor method)

- Instance의 reference counter가 0(zero) 이 될 때 자동으로 호출
- 메모리 해체 등의 종료작업을 위함
- 특수 method '__del__()'로 정의

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

    def __del__(self):
        print("Class is deleted!")
```

__del__() method 정의

# 소멸자 메소드 (destructor method)

- https://docs.python.org/3/library/sys.html

- sys.getrefcount(object)
  - Return the reference count of the object
  - The count returned is generally one higher than you might expect, because it includes the (temporary) reference as an argument to getrefcount()

Previous topic
Python Runtime Services

Next topic
**sys.monitoring** — Execution event monitoring

This Page
Report a Bug
Show Source

are used.

The filesystem encoding and error handler are configured at Python startup by the `PyConfig_Read()` function: see `filesystem_encoding` and `filesystem_errors` members of `PyConfig`.

*New in version 3.6.*

### sys.get_int_max_str_digits()

Returns the current value for the integer string conversion length limitation. See also `set_int_max_str_digits()`.

*New in version 3.11.*

### sys.getrefcount(*object*)

Return the reference count of the *object*. The count returned is generally one higher than you might expect, because it includes the (temporary) reference as an argument to `getrefcount()`.

Note that the returned value may not actually reflect how many references to the object are actually held. For example, some objects are “immortal” and have a very high refcount that does not reflect the actual number of references. Consequently, do not rely on the returned value to be accurate, other than a value of 0 or 1.

*Changed in version 3.12:* Immortal objects have very large refcounts that do not match the actual number of references to the object.

### sys.getrecursionlimit()

Return the current value of the recursion limit, the maximum depth of the Python interpreter stack. This limit prevents infinite recursion from causing an overflow of the C stack and crashing Python. It can be set by `setrecursionlimit()`.

### sys.getsizeof(*object*[, *default*])

Return the size of an object in bytes. The object can be any type of object. All built-in objects will return correct results, but this does not have to hold true for third-party extensions as it is implementation specific.

Only the memory consumption directly attributed to the object is accounted for, not the memory consumption of objects it refers to.

If given, *default* will be returned if the object does not provide means to retrieve the size. Otherwise a `TypeError` will be raised.

Destructor.py

File Edit Format Run Options Window Help

```python
# Adding an __del__() method

import sys

class Ball:

    #member 변수
    color = "없음"
    size = 0
    direction = "down"

    def __init__(self, color, size, direction):     # 생성자(constructor)
        print("₩ni-1) 생성자 __init__() called.")

        self.color = color
        self.size = size
        self.direction = direction
        print("i-2) self.color = ",color)
        print("i-3) self.size = ",size)
        print("i-4) self.direction = ",direction)

    def __del__(self):                              # 소멸자(destructor)
        print("₩nd-1) 소멸자. Object is deleted !")

    def bounce(self):
        print("b-1) in bounce")
        if self.direction == "down":
            self.direction = "up"
            print("b-1) self.direction = ", self.direction)

# main# Garbage Collection
# 객체가 참조될 때마다 reference count가 1 증가
# 참조가 해제될 때 마다 reference count가 1 감소
# 카운트가 0 이 되면 객체는 메모리 공간에서 삭제

print("1) main ")
print("₩n2) myBall 객체 생성")
myBall = Ball("red", "small", "down")   # myBall instance 생성
                                        # reference counter 증가 : 1
print("₩n3) myBall의 reference count = ", sys.getrefcount(myBall))
print("    myBall.size = ", myBall.size)
print("    myBall.color = ", myBall.color)
print("    myBall.direction = ", myBall.direction)
print("4) myBall의 reference count = ", sys.getrefcount(myBall))
```

```
1) main

2) myBall 객체 생성

i-1) 생성자 __init__() called.
i-2) self.color =  red
i-3) self.size =  small
i-4) self.direction =  down

3) myBall의 reference count =  2
   myBall.size =  small
   myBall.color =  red
   myBall.direction =  down
4) myBall의 reference count =  2

5) call myBall.bounce()
b-1) in bounce
b-1) self.direction =  up

6) myBall.direction =  up
7) myBall의 reference count =  2

8) myBall의 reference count =  3
9) myBall의 reference count =  4

10) myBall의 reference count =  3
11) myBall의 reference count =  2
12) myBall 객체에 대한 참조를 모두 해제
13) myBall의 reference count =  2
14) myBall의 size, color, direction 값들이  class member variable 초기값으로 초기화
    myBall.size =  0
    myBall.color =  없음
    myBall.direction =  down
15) myBall의 reference count =  2

d-1) 소멸자. Object is deleted !
16) 소멸자가 자동으로 불리어져서 myBall 객체가 메모리에서 사라짐.
Traceback (most recent call last):
  File "C:₩소사₩강의예제₩Destructor.py", line 78, in <module>
    print("17) myBall의 reference count = ", sys.getrefcount(myBall))
NameError: name 'myBall' is not defined. Did you mean: 'Ball'?
```

```
45
46 print("₩n5) call myBall.bounce()")
47 myBall.bounce()
48 print("₩n6) myBall.direction = ", myBall.direction)
49 print("7) myBall의 reference count = ", sys.getrefcount(myBall))
50
51 # reference count 증가 예
52 yourBall = myBall
53 print("₩n8) myBall의 reference count = ", sys.getrefcount(myBall))
54
55 herBall = myBall
56 print("9) myBall의 reference count = ", sys.getrefcount(myBall))
57
58 del yourBall
59 print("₩n10) myBall의 reference count = ", sys.getrefcount(myBall))
60
61 del herBall
62 print("11) myBall의 reference count = ", sys.getrefcount(myBall))
63
64 # myBall 객체에 대한 참조를 모두 해제
65 print("12) myBall 객체에 대한 참조를 모두 해제")
66
67 del myBall.size, myBall.color, myBall.direction
68 print("13) myBall의 reference count = ", sys.getrefcount(myBall))
69
70 print("14) myBall의 size, color, direction 값들이  class member variable 초기값으로 초기화")
71 print("    myBall.size = ", myBall.size)
72 print("    myBall.color = ", myBall.color)
73 print("    myBall.direction = ", myBall.direction)
74 print("15) myBall의 reference count = ", sys.getrefcount(myBall))
75
76 del myBall
77 print("16) 소멸자가 자동으로 불리어져서 myBall 객체가 메모리에서 사라짐.")
78 print("17) myBall의 reference count = ", sys.getrefcount(myBall))
79
```

```
1) main

2) myBall 객체 생성

i-1) 생성자 __init__() called.
i-2) self.color =  red
i-3) self.size =  small
i-4) self.direction =  down

3) myBall의 reference count =  2
   myBall.size =  small
   myBall.color =  red
   myBall.direction =  down
4) myBall의 reference count =  2

5) call myBall.bounce()
b-1) in bounce
b-1) self.direction =  up

6) myBall.direction =  up
7) myBall의 reference count =  2

8) myBall의 reference count =  3
9) myBall의 reference count =  4

10) myBall의 reference count =  3
11) myBall의 reference count =  2
12) myBall 객체에 대한 참조를 모두 해제
13) myBall의 reference count =  2
14) myBall의 size, color, direction 값들이  class member variable 초기값으로 초기화
    myBall.size =  0
    myBall.color =  없음
    myBall.direction =  down
15) myBall의 reference count =  2

d-1) 소멸자. Object is deleted !
16) 소멸자가 자동으로 불리어져서 myBall 객체가 메모리에서 사라짐.
Traceback (most recent call last):
  File "C:₩소사₩강의예제₩Destructor.py", line 78, in <module>
    print("17) myBall의 reference count = ", sys.getrefcount(myBall))
NameError: name 'myBall' is not defined. Did you mean: 'Ball'?
```

# Method 정의

- Method는 Class 안에 정의된 함수이므로 함수를 정의하는 것과 아주 유사함.
  하지만 첫 번째 매개변수는 항상 self이어야 함.

```
class Television:
        def __init__(self, channel, volume, on):
                self.channel = channel
                self.volume = volume
                self.on = on

        def show(self):
                print(self.channel, self.volume, self.on)

        def setChannel(self, channel):
                self.channel = channel

        def getChannel(self):
                return self.channel
```

# self

- 하나의 class에서 여러 instance를 생성 가능
- 현재의 instance를 가리키는 기능을 하는 지시어. 즉 method가 어떤 instance에서 호출됐는지 알려주는 instance 참조자 (instance reference)

```
class Ball:
    def bounce(self):
        if self.direction == "down":
            self.direction = "up"

myBall =  Ball()
yourBall =  Ball()
```

- 위의 예에서 bounce()라는 method입장에서는 어느 instance가 자신을 호출했는 지 알아야 함
- self 인자는 어느 객체가 method를 호출했는 지 알려줌

→ 이를 instance reference라 함.

Object이 method를 호출할 때 어떤 instance가 호출했는지 instance reference를 method로 자동으로 넘겨줌

Self.py

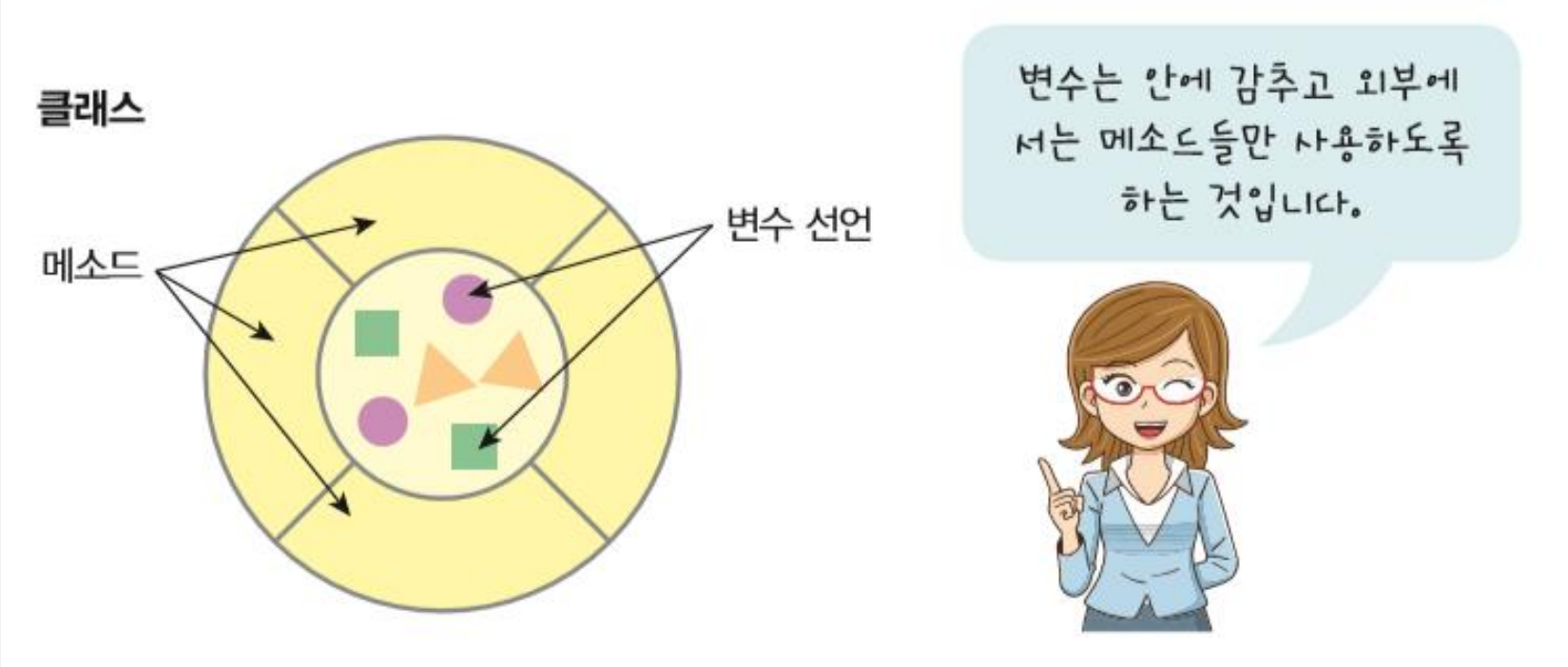
File Edit Format Run Options Window Help

```
class Ball:

    size = 0
    direction = "default"

    def __init__(self, in_size, in_direction) :
        print("i-1) In init, self =", self)
        self.size = in_size
        self.direction = in_direction

        print("i-2) self.size =", self.size)
        print("i-3) self.dirction =", self.direction,"₩n")

    def bounce(self):
        print("b-1) self = ", self)
        if self.direction == "down":
            self.direction = "up"

# main
print("1) main ₩n")

print("2) myBall 생성")
myBall = Ball(10, "down")

print("3) yourBall 생성")
yourBall = Ball(20, "up")

print("4) call myBall.bounce()")
myBall.bounce()

print("₩n5) call yourBall.bounce()")
yourBall.bounce()

```

```
1) main

2) myBall 생성
i-1) In init, self = <__main__.Ball object at 0x000001D8E8EC6970>
i-2) self.size = 10
i-3) self.dirction = down

3) yourBall 생성
i-1) In init, self = <__main__.Ball object at 0x000001D8E8EC6C40>
i-2) self.size = 20
i-3) self.dirction = up

4) call myBall.bounce()
b-1) self =  <__main__.Ball object at 0x000001D8E8EC6970>

5) call yourBall.bounce()
b-1) self =  <__main__.Ball object at 0x000001D8E8EC6C40>
>>>
```

# 정보은닉(Information Hiding)

- 구현의 세부 사항을 Class 안에 감추는 것



# Private instance variable

- Instance variable을 private으로 선언하여 사용하는 것을 권장
  - 선언하고자 하는 변수 앞에 __ (underline 2개)를 붙이면 private으로 선언됨
- Method도 이름 앞에 __를 붙이면 private method가 됨

```
class Student:
        def __init__(self, name=None, age=0):
                self.__name = name
                self.__age = age

obj = Student()
print(obj.__age)
```

```
...
AttributeError: 'Student' object has no attribute '__age'
```

student.py

File Edit Format Run Options Window Help

```
class Student:
    def __init__(self, name=None, age=0):
        print("₩ni-1) In init, name과 age를 private 선언")
        self.__name = name
        self.__age = age

        print("i-2) self.__name  =", self.__name)
        print("       self.__age  =", self.__age)

#main
print("1) main start")
print("2) 객체 s1 생성")
s1 = Student("Park", 20)

print("₩n3) private 선언된 __age 값을 직접 접근시 error 발생")
print(s1.__age)
```

```
1) main start
2) 객체 s1 생성

i-1) In init, name과 age를 private 선언
i-2) self.__name  = Park
       self.__age  = 20

3) private 선언된 __age 값을 직접 접근시 error 발생
Traceback (most recent call last):
  File "C:₩소사₩강의예제₩student.py", line 16, in <module>
    print(s1.__age)
AttributeError: 'Student' object has no attribute '__age'
```

# 접근자와 설정자

- Private 선언된 변수(attribute)를 읽어(read)오고 변수값을 변경(write)하는 방법

→ 접근자와 설정자에게 위탁(delegation)

  - 접근자(getter, accessor) : instance 변수값 (attributes)을 반환 (**read only**)
  - 설정자(setter, mutator) : instance 변수값(attributes)을 설정(변경) **(write)**

student_with_GetterSetter.py

File Edit Format Run Options Window Help

```
1  class Student:
2      def __init__(self, name=None, age=0):
3          print("₩ni-1) name과 age를 private 선언")
4          self.__name = name
5          self.__age = age
6          print("i-2) self.__name =", self.__name)
7          print("       self.__age =", self.__age)
8
9      def getName(self): # getter for __name
10         print("₩ngetName) self.__name =", self.__name)
11         return self.__name
12
13     def getAge(self): #getter for __age
14         print("₩ngetAge) self.__age =", self.__age)
15         return self.__age
16
17     def setName(self, name): # setter for __name
18         self.__name= name
19         print("₩nsetName) self.__name =", self.__name)
20
21     def setAge(self, age): # setter for __age
22         self.__age=age
23         print("setAge) self.__age =", self.__age)
24
25 #main
26 print("1) main start")
27 print("2) 객체 s1 생성")
28 s1 = Student("Park", 20)
29
30 # error !!
31 #print(s1.__age)
32
33 print("3) s1.__name =", s1.getName())
34 print("4) s1.__age =",s1.getAge())
35
36 print("₩n5) s1 attribute  __name, __age 값 변경")
37 s1.setName("Hyun")
38 s1.setAge(25)
39
40 print("6) s1.__name =", s1.getName())
41 print("7) s1.__age =",s1.getAge())
42
```

```
1) main start
2) 객체 s1 생성

i-1) name과 age를 private 선언
i-2) self.__name = Park
       self.__age = 20

getName) self.__name = Park
3) s1.__name = Park

getAge) self.__age = 20
4) s1.__age = 20

5) s1 attribute  __name, __age 값 변경

setName) self.__name = Hyun
setAge) self.__age = 25

getName) self.__name = Hyun
6) s1.__name = Hyun

getAge) self.__age = 25
7) s1.__age = 25
```

television_class_getter_setter.py -

File Edit Format Run Options Window Help

```python
class Television:

    __channel  = 0
    __volume  = 0
    __on = True

    def __init__(self, in_channel, in_volume, in_on):
        print("init-1) In init")
        self.__channel = in_channel
        self.__volume = in_volume
        self.__on = in_on
        print("init-2) self.__channel  =", self.__channel)
        print("        self.__volume  =", self.__volume)
        print("        self.__on  =", self.__on)

    # getters
    def getChannel(self):
        print("getChannel) self.__channel =", self.__channel)
        return self.__channel

    def getVolume(self):
        print("getVolume) self.__volume =", self.__volume)
        return self.__volume

    def getOn(self):
        print("getOn) self.__On =", self.__on)
        return self.__on

    #setters
    def setChannel(self, in_channel):
        self.__channel = in_channel
        print("setChannel) self.__channel =", self.__channel)

    def setVolume(self, in_volume):
        self.__volume = in_volume
        print("setVolumw) self.__volume =", self.__volume)

    def setOn(self, in_on):
        self.__on = in_on
        print("setOn) self.__On =", self.__on)

    # show all current values of atttibutes
    def show(self):
        print("show) self.__channel  =", self.__channel)
        print("      self.__volume  =", self.__volume)
        print("      self.__on  =", self.__on)


class Main: # driver class

    def main(self):

        print("1) main start")
        print("Wn2) 객체 t 생성")
        t = Television(9, 10, True)

        # print("t.__channel", t.__channel)  # error
        t.show()

        print("Wn3) 채널변경")
        t.setChannel(11)
        t.getChannel()

        print("Wn4) 볼륨 변경")
        t.setVolume(20)
        t.getVolume()

        print("Wn5) 전원끄기")
        t.setOn(False)
        t.getOn()

# triggering
m = Main()
m.main()
```

```
1) main start

2) 객체 t 생성
init-1) In init
init-2) self.__channel  = 9
        self.__volume  = 10
        self.__on  = True
show) self.__channel  = 9
      self.__volume  = 10
      self.__on  = True

3) 채널변경
setChannel) self.__channel = 11
getChannel) self.__channel = 11

4) 볼륨 변경
setVolumw) self.__volume = 20
getVolume) self.__volume = 20

5) 전원끄기
setOn) self.__On = False
getOn) self.__On = False
```

# Lab: 원을 Class로 표현

- 원을 Class도 표시해보자.
- 원은 반지름(radius)을 가지고 있다.
- 원의 넓이와 둘레를 계산하는 Method도 정의해보자.
  설정자와 접근자 Method도 작성한다.

circle_class.py

File Edit Format Run Options Window Help

```python
import math

class Circle:

    def __init__(self, radius):

        if radius <= 0:
            print("Circleinit-1) 반지름 값이 0 보다 작습니다.")
            exit()
        else:
            self.__radius = radius
            self.__Pie = math.pi
            print("Circleinit-2) self.__radius =", self.__radius)
            print("                self.__Pie =", self.__Pie)

    def setRadius(self, r):
        self.__radius = r

    def getRadius(self):
        return self.__radius

    def calcArea(self):
        area = self.__Pie * self.__radius * self.__radius
        return area

    def calcCircum(self):
        circumference = 2.0 * self.__Pie * self.__radius
        return circumference
```

```python
class Radius_input:

    def __init__(self, rad = 1.0):
        self.__input_radius = rad
        print("Ra_init) self.__input_radius =", self.__input_radius)

    def in_value(self):
        self.__r = input("₩n>>> 원의 반지름을 입력하시오. 기본값 (1) : ")

        if self.__r == "":
            print("Ra_in_value-1) default self.__input_radius =", self.__input_radius)

        else :
            self.__input_radius = float(self.__r)
            print("Ra_in_value-2)  self.__input_radius =", self.__input_radius)

        return self.__input_radius

class Main: # driver

    __r = 0

    def main(self):
        print("1) main")
        rad = Radius_input()

        self.__r = rad.in_value()
        print("₩n2) self.__r =", self.__r)

        c1 = Circle(self.__r)

        print("₩n3) 원의 반지름", c1.getRadius())
        print("4) 원의 넓이=", c1.calcArea())
        print("5) 원의 둘레=", c1.calcCircum())

        print("6) setRadius() 이용")
        c1.setRadius(10)
        print("₩n7) 원의 반지름=", c1.getRadius())
        print("8) 원의 넓이=", c1.calcArea())
        print("9) 원의 둘레=", c1.calcCircum())

# Ignition
m = Main()
m.main()
```

```
1) main
Ra_init) self.__input_radius = 1.0

>>> 원의 반지름을 입력하시오. 기본값 (1) : 3
Ra_in_value-2)  self.__input_radius = 3.0

2) self.__r = 3.0
Circleinit-2) self.__radius = 3.0
                self.__Pie = 3.141592653589793

3) 원의 반지름 3.0
4) 원의 넓이= 28.274333882308138
5) 원의 둘레= 18.84955592153876
6) setRadius() 이용

7) 원의 반지름= 10
8) 원의 넓이= 314.1592653589793
9) 원의 둘레= 62.83185307179586
```

# Lab: 은행 계좌

- 우리는 은행 계좌에 돈을 저금할 수 있고 인출할 수도 있다. 은행 계좌를 Class로 모델링하여 보자. 은행 계좌는 현재 잔액(balance)만을 인스턴스 변수로 가진다. 생성자와 인출 Method withdraw()와 저축 Method deposit() 만을 가정하자.

```
통장에서   100 가 출금되었음
통장에   10 가 입금되었음
```

# Solution

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def withdraw(self, amount):
        self.__balance -= amount
        print("통장에서 ", amount, "가 출금되었음" )
        return self.__balance

    def deposit(self, amount):
        self.__balance += amount
        print( " 통장에  " , amount,  " 가 입금되었음")
        return self.__balance

bank = BankAccount()
bank.deposit(100)
bank.withdraw(10)
```

# Lab: 고양이 Class

- 고양이를 Class로 정의. 고양이는 이름(name)과 나이(age)를 속성으로 가진다.


```
Missy 3
Lucky 5
```

# Solution

```
class Cat:
   def __init__(self, name, age):
      self.__name = name
      self.__age = age

   def setName(self, name):
      self.__name = name

   def getName(self):
      return self.__name

   def setAge(self, age):
      self.__age = age

   def getAge(self):
      return self.__age


missy = Cat('Missy', 3)
lucky = Cat('Lucky', 5)

print (missy.getName(), missy.getAge())
print (lucky.getName(), lucky.getAge())
```

# Lab: 객체 생성과 사용

- 상자를 나타내는 Box Class를 작성하여 보자. Box Class는 가로길이, 세로길이, 높이를 나타내는 인스턴스 변수를 가진다.

```
(100, 100, 100)
상자의 부피는  1000000
```

# Solution

```
class Box:
    def __init__(self, width=0, length=0, height=0):
        self.__width = width
        self.__length = length
        self.__height = height

    def setWidth(self, width):
        self.__width = width;

    def setLength(self, length):
        self.__length = length;

    def setHeight(self, height):
        self.__height = height;

    def getVolume(self):
        return self.__width*self.__length*self.__height

    def __str__(self):
        return '(%d, %d, %d)' % (self.__width, self.__length,
self.__height)

# main
box = Box(100, 100, 100)
print(box)
print('상자의 부피는 ', box.getVolume())
```

# Lab: 자동차 Class 작성

- 자동차를 나타내는 Class를 정의하여 보자. 예를 들어, 자동차 객체의 경우, 속성은 색상, 현재 속도, 현재 기어 등이다. 자동차의 동작은 기아 변속하기, 가속하기, 감속하기 등을 들 수 있다. 이 중에서 다음 그림과 같은 속성과 동작만을 추려서 구현해 보자.

```
(100, 3, white)
```

# Solution

```
class Car:
    def __init__(self, speed=0, gear=1, color="white"):
        self.__speed = speed
        self.__gear = gear
        self.__color = color

    def setSpeed(self, speed):
        self.__speed = speed;

    def setGear(self, gear):
        self.__gear = gear;

    def setColor(self, color):
        self.__color = color;

    def __str__(self):
        return '(%d, %d, %s)' % (self.__speed, self.__gear, self.__color)

myCar = Car()
myCar.setGear(3);
myCar.setSpeed(100);
print(myCar)
```

# 객체를 함수로 전달할 때

- 객체를 argument로 지정하여 함수로 전달할 수 있음
  - User-defined 객체가 전달되면 함수가 이 객체를 변경할 수 있음
  - 문자열 등과 같은 system-defined 객체의 경우 이를 전달받은 함수는 이 객체를 변경할 수 없음

# 객체를 함수로 전달할 때

rectangle.py

File Edit Format Run Options Window Help

```
1  # 사각형을 클래스로 정의
2  count = 5
3
4  class Rectangle:
5          def __init__(self, side = 0):
6                  self.side = side
7                  print("init) self.side =", self.side)
8
9          def getArea(self):
10                 return self.side*self.side
11
12 # 사각형 객체(r)와 반복횟수(n)를 받아서 변을 증가시키면서 면적을 출력
13 def printAreas(r, n):
14         print("₩nprintArea) 반복횟수 =", n)
15         while n >= 1:
16                 print(">> r.side =", r.side, "₩t", "area = ", r.getArea())
17                 r.side = r.side + 1
18                 n = n - 1
19
20 # printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인한다.
21 print("1) main")
22 myRect = Rectangle();
23
24 printAreas(myRect, count)
```

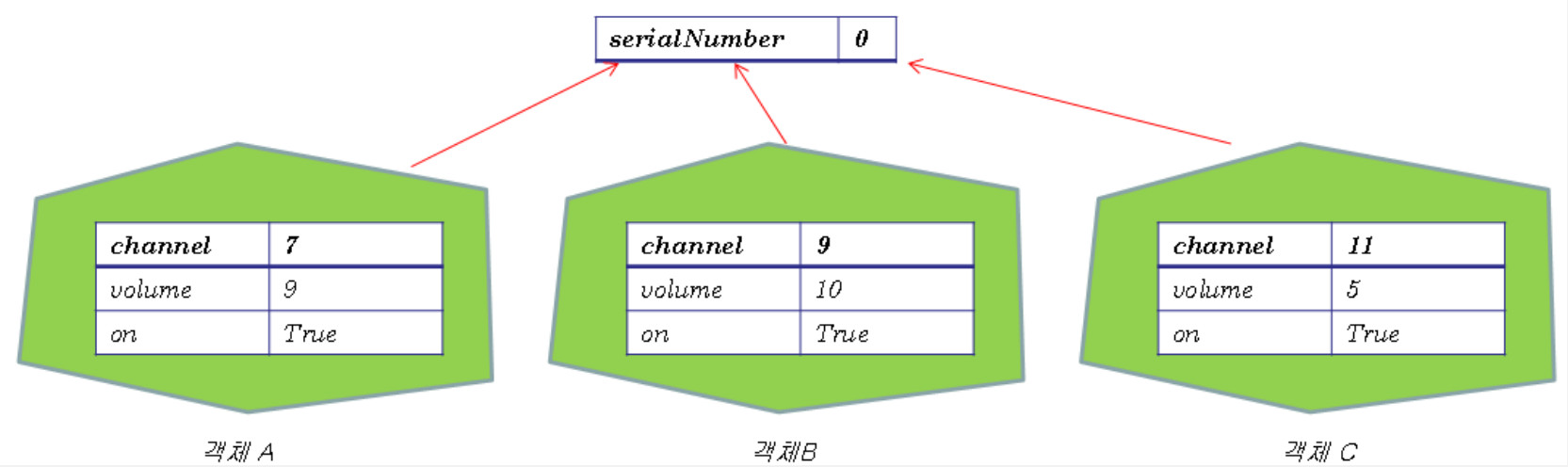
```
1) main
init) self.side = 0

printArea) 반복횟수 = 5
>> r.side = 0        area =  0
>> r.side = 1        area =  1
>> r.side = 2        area =  4
>> r.side = 3        area =  9
>> r.side = 4        area =  16
```

# Class 변수(static 변수)

- 하나의 class에서 생성된 (여러) object(instance)들이 상호공유하는 변수
- 이들 변수는 모든 object(instance)를 통틀어서 unique해야 함. 모든 객체가 이것을 공유
- 이러한 변수를 정적변수(static member, class member, static variable)라고 함

# Class 변수(static 변수)

```
class Television:
        serialNumber = 0          # 이것이 정적변수이다.
        def __init__(self):
                Television.serialNumber += 1
        self.number = Television.serialNumber
                ...
```



television_class_variable.py

File Edit Format Run Options Window Help

```python
class Television:

    count = 0                # class variable
    serial_number = 100   # class variable

    def __init__(self, channel, volume, on):
        self.channel = channel
        self.volume = volume
        self.on = on

        Television.count += 1
        print("init) Television.count =", Television.count)

        Television.serial_number += 10
        print("init) Television.serial_number =", Television.serial_number)

    def __getCount__(self):
#        print("g-1) self.count =", self.count)
        return self.count

    def __getSerial_number__(self):
        print("g-1) self.serial_number =", self.serial_number)
        return self.serial_number

#main
print("1) 객체 t1 생성")
t1 = Television(9, 10, True)
print("₩n2) Television.count =", Television.count)
print("    Television.serial_number =", Television.serial_number)

print("₩n3) 객체 t2 생성")
t2 = Television(11, 12, True)
print("₩n4) Television.count =", Television.count)
print("    Television.serial_number =", Television.serial_number)

print("₩n5) 객체 t3 생성")
t3 = Television(13, 15, False)
print("₩n6) Television.count =", Television.count)
print("   Television.serial_number =", Television.serial_number)

print("₩n7) Class 변수 serial_number 값을 200으로 변경")
Television.serial_number = 200
print("8) Television.serial_number = ", Television.serial_number)

print("₩n9) 객체 t4 생성")
t4 = Television(12, 20, False)
print("₩n10) Television.count =", Television.count)
print("    Television.serial_number =", Television.serial_number)

print("₩n11) t1.count =", t1.__getCount__())
print("12) t2.count =", t2.__getCount__())
print("13) t3.count =", t3.__getCount__())
print("14) t4.count =", t4.__getCount__())

print("₩n15) t1.count =", t1.count)
print("    t2.count =", t2.count)
print("    t3.count =", t3.count)
print("    t4.count =", t4.count)
```

```
1) 객체 t1 생성
init) Television.count = 1
init) Television.serial_number = 110

2) Television.count = 1
   Television.serial_number = 110

3) 객체 t2 생성
init) Television.count = 2
init) Television.serial_number = 120

4) Television.count = 2
   Television.serial_number = 120

5) 객체 t3 생성
init) Television.count = 3
init) Television.serial_number = 130

6) Television.count = 3
  Television.serial_number = 130

7) Class 변수 serial_number 값을 200으로 변경
8) Television.serial_number =  200

9) 객체 t4 생성
init) Television.count = 4
init) Television.serial_number = 210

10) Television.count = 4
   Television.serial_number = 210

11) t1.count = 4
12) t2.count = 4
13) t3.count = 4
14) t4.count = 4

15) t1.count = 4
   t2.count = 4
   t3.count = 4
   t4.count = 4
>>>
```

# Special method : 객체 출력 method

- Special method
  - class를 정의 시 Python이 자동으로 포함시키는 method들
- 그 중의 하나가 특수 method '__str__()'
  - 객체를 print로 출력할 때 Python이 어떤 내용으로 표시할 지 알려줌
  - 기본출력 내용
    - Instance가 정의된 곳 (앞의 예제에서는 __main__ )
    - Class 이름 (앞의 예제에서는 Ball)
    - Instance가 저장되어있는 메모리상의 위치

# Default __str__()

- Special method
  - class를 정의 시 Python이 자동으로 포함시키는 method들
- 그 중의 하나가 특수 method '__str__()'
  - 객체를 print로 출력할 때 Python이 어떤 내용으로 표시할 지 알려줌
  - 기본출력 내용
    - Instance가 정의된 곳 (앞의 예제에서는 __main__ )
    - Class 이름 (앞의 예제에서는 Ball)
    - Instance가 저장되어있는 메모리상의 위치

default__str__.py

File Edit Format Run Options Window Help

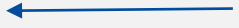
```
1  # default __str__()를 이용하여 객체에 대한 기본정보를 출력
2
3  class Ball:
4      def __init__(self, color, size, direction):
5          print("₩ninit) self =", self)
6          self.color = color
7          self.size = size
8          self.direction = direction
9
10 #main
11 myBall = Ball("red", "small", "down")
12 print (myBall)
13
14 yourBall = Ball("blue", "large", "up")
15 print (yourBall)
```

```
init) self = <__main__.Ball object at 0x000002452DBA8910>
<__main__.Ball object at 0x000002452DBA8910>

init) self = <__main__.Ball object at 0x000002452DC36FD0>
<__main__.Ball object at 0x000002452DC36FD0>
```

# Special method : 객체 출력 method

- 특수 method '__str__()'
  - Default로 보여주는 내용 이외에 개발자가 객체에 대해 정의한 내용으로도 출력가능

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

    def __str__(self):
        msg = "Hi, I'm a " + self.size + " " + self.color + " ball!"
        return msg

myBall = Ball("red", "small", "down")
print(myBall)
```



__str__() method에서 반환되는 문자열 출력

# Special method : 객체 출력 method

user_defined__Str__.py

File Edit Format Run Options Window Help

```
1  # Using __str__() to change how the object prints
2
3  class Ball:
4
5      def __init__(self, color, size, direction):
6          self.color = color
7          self.size = size
8          self.direction = direction
9
10     # here's the special method __str__()
11     def __str__(self):
12         msg = "Hi, I'm a " + self.size + " " + self.color + " ball!"
13         print("s-1) msg =", msg)
14
15         return msg
16
17 # main
18 myBall = Ball("red", "small", "down")
19 print (myBall)
```

```
s-1) msg = Hi, I'm a small red ball!
Hi, I'm a small red ball!
>>>
```

Listing_14-6.py

File Edit Format Run Options Window Help

```python
class HotDog:

    def __init__(self):
        self.cooked_level = 0
        self.cooked_string = "Raw"
        self.condiments = []

        print("Wni-1) self.cooked_level =",self.cooked_level)
        print("     self.cooked_string =",self.cooked_string)
        print("     self.condiments  =",self.condiments)

    def __str__(self):
        msg = "hot dog"

        if len(self.condiments) > 0:
            msg = msg + " with "

        for i in self.condiments:
            msg = msg + i + ", "
            print("s-1) msg =", msg)

        msg = msg.strip(", ") # 마지막 "," 삭제
        print("s-2) msg =", msg)

        msg = self.cooked_string + " " + msg + "."
        print("s-3) 생성된 메시지 :", msg)
        return msg

    def cook(self, time):
        print("c-1) self.cooked_level = {}, time = {}" .format(self.cooked_level, time))
        self.cooked_level = self.cooked_level + time
        print("c-2) in cook(), 요청된 굽는 시간 :", self.cooked_level)

        if self.cooked_level > 8:
            self.cooked_string = "Charcoal"
        elif self.cooked_level > 5:
            self.cooked_string = "Well-done"
        elif self.cooked_level > 3:
            self.cooked_string = "Medium"
        else:
            self.cooked_string = "Raw"

        print("c-3) in cook(), cooked_string = ", self.cooked_string)

    def addCondiment(self, stuff):
        print("Wna-1) 추가된 양념 = ", stuff)
        self.condiments.append(stuff)
        print("a-2) 최종 양념 = ", self.condiments)

```

```
1) main start.myDoc객체생성

i-1) self.cooked_level = 0
     self.cooked_string = Raw
     self.condiments  = []

2) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Raw hot dog.
Raw hot dog.

3) Cooking hot dog for 4 minutes.
c-1) self.cooked_level = 0, time = 4
c-2) in cook(), 요청된 굽는 시간 : 4
c-3) in cook(), cooked_string =  Medium

4) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Medium hot dog.
Medium hot dog.

5) Cooking hot dog for 3 more minutes.
c-1) self.cooked_level = 4, time = 3
c-2) in cook(), 요청된 굽는 시간 : 7
c-3) in cook(), cooked_string =  Well-done

6) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Well-done hot dog.
Well-done hot dog.

7) What happens if I cook it for 10 more minutes?
c-1) self.cooked_level = 7, time = 10
c-2) in cook(), 요청된 굽는 시간 : 17
c-3) in cook(), cooked_string =  Charcoal

8) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Charcoal hot dog.
Charcoal hot dog.

9) Now, I'm going to add some stuff on my hot dog

a-1) 추가된 양념 =  ketchup
a-2) 최종 양념 =  ['ketchup']

a-1) 추가된 양념 =  mustard
a-2) 최종 양념 =  ['ketchup', 'mustard']
```

```
10) myDog 객체 정보 출력
s-1) msg = hot dog with ketchup,
s-1) msg = hot dog with ketchup, mustard,
s-2) msg = hot dog with ketchup, mustard
s-3) 생성된 메시지 : Charcoal hot dog with ketchup, mustard.
Charcoal hot dog with ketchup, mustard.
```

```python
# main
print("1) main start.myDoc객체생성")
myDog = HotDog()
print("₩n2) myDog 객체 정보 출력")
print(myDog)

print("₩n3) Cooking hot dog for 4 minutes.")
myDog.cook(4)

print("₩n4) myDog 객체 정보 출력")
print (myDog)

print ("₩n5) Cooking hot dog for 3 more minutes.")
myDog.cook(3)

print ("₩n6) myDog 객체 정보 출력")
print (myDog)

print ("₩n7) What happens if I cook it for 10 more minutes?")
myDog.cook(10)
print ("₩n8) myDog 객체 정보 출력")
print (myDog)

print ("₩n9) Now, I'm going to add some stuff on my hot dog")
myDog.addCondiment("ketchup")
myDog.addCondiment("mustard")

print ("₩n10) myDog 객체 정보 출력")
print (myDog)

```

```
1) main start.myDoc객체생성

i-1) self.cooked_level = 0
     self.cooked_string = Raw
     self.condiments  = []

2) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Raw hot dog.
Raw hot dog.

3) Cooking hot dog for 4 minutes.
c-1) self.cooked_level = 0, time = 4
c-2) in cook(), 요청된 굽는 시간 : 4
c-3) in cook(), cooked_string =  Medium

4) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Medium hot dog.
Medium hot dog.

5) Cooking hot dog for 3 more minutes.
c-1) self.cooked_level = 4, time = 3
c-2) in cook(), 요청된 굽는 시간 : 7
c-3) in cook(), cooked_string =  Well-done

6) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Well-done hot dog.
Well-done hot dog.

7) What happens if I cook it for 10 more minutes?
c-1) self.cooked_level = 7, time = 10
c-2) in cook(), 요청된 굽는 시간 : 17
c-3) in cook(), cooked_string =  Charcoal

8) myDog 객체 정보 출력
s-2) msg = hot dog
s-3) 생성된 메시지 : Charcoal hot dog.
Charcoal hot dog.

9) Now, I'm going to add some stuff on my hot dog

a-1) 추가된 양념 =  ketchup
a-2) 최종 양념 =  ['ketchup']

a-1) 추가된 양념 =  mustard
a-2) 최종 양념 =  ['ketchup', 'mustard']

10) myDog 객체 정보 출력
s-1) msg = hot dog with ketchup,
s-1) msg = hot dog with ketchup, mustard,
s-2) msg = hot dog with ketchup, mustard
s-3) 생성된 메시지 : Charcoal hot dog with ketchup, mustard.
Charcoal hot dog with ketchup, mustard.
```

# Special method : 연산자 관련 method

- Python에는 연산자operator(+, -, *, /)에 관련된 **특수 메소드(special method)**가 있음

```
class Circle:
        ...
        def __eq__(self, other):
                return self.radius == other.radius

c1 = Circle(10)
c2 = Circle(10)
if c1 == c2:
        print("원의 반지름은 동일합니다. ")
```

# Special method

| 연산자 | 메소드 | 설명 |
|---|---|---|
| x + y | __add__(self, y) | 덧셈 |
| x - y | __sub__(self, y) | 뺄셈 |
| x * y | __mul__(self, y) | 곱셈 |
| x / y | __truediv__(self, y) | 실수나눗셈 |
| x // y | __floordiv__(self, y) | 정수나눗셈 |
| x % y | __mod__(self, y) | 나머지 |
| divmod(x, y) | __divmod__(self, y) | 실수나눗셈과 나머지 |
| x ** y | __pow__(self, y) | 지수 |
| x << y | __lshift__(self, y) | 왼쪽 비트 이동 |
| x >> y | __rshift__(self, y) | 오른쪽 비트 이동 |
| x <= y | __le__(self, y) | less than or equal(작거나 같다) |
| x < y | __lt__(self, y) | less than(작다) |
| x >= y | __ge__(self, y) | greater than or equal(크거나 같다) |
| x > y | __gt__(self, y) | greater than(크다) |
| x == y | __eq__(self, y) | 같다 |
| x != y | __neq__(self, y) | 같지않다 |

# Special method : 연산자 관련 method

- __add__()

  __add__() 연산을 2개 vector 값의 합을 구하는 method로 재정의

  - 2차원 공간에서 vector는 (a, b)와 같이 2개의 실수로 표현
  - +

    (a, b) + (c, d) = (a + c, b + d)
  - -

    (a, b) - (c, d) = (a - c, b - d)

# 예제

```
class Vector2D :
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)

u = Vector2D(0,1)
v = Vector2D(1,0)
w = Vector2D(1,1)
a = u + v
print( a)
```

# 예제 – vector operator overload

vector.py

File Edit Format Run Options Window Help

```
1  class Vector2D :
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5          print("init) self.x = ", self.x, "self.y = ", self.y)
6
7      def __add__(self, other):
8          print("add) self.x = ", self.x, "self.y = ", self.y)
9          print("add) other.x = ", other.x, "other.y = ", other.y)
10         return Vector2D(self.x + other.x, self.y + other.y)
11
12     def __sub__(self, other):
13         print("sub) self.x = ", self.x, "self.y = ", self.y)
14         print("sub) other.x = ", other.x, "other.y = ", other.y)
15         return Vector2D(self.x - other.x, self.y - other.y)
16
17     def __eq__(self, other):
18         print("equ) self.x = ", self.x, "self.y = ", self.y)
19         print("equ) other.x = ", other.x, "other.y = ", other.y)
20         return self.x == other.x and self.y == other.y
21
22     def __str__(self):
23         return '(%g, %g)' % (self.x, self.y)
24
```

```
25 #main
26 print("main) 객체 u 생성")
27 u = Vector2D(0,1)
28 print("1) print u = ", u, "₩n")
29
30 print("2) 객체 v 생성")
31 v = Vector2D(1,0)
32 print("3) print v = ", v,"₩n")
33
34 print("4) 객체 w 생성")
35 w = Vector2D(1,1)
36 print("5) print w = ", w,"₩n")
37
38 print("6) 객체 add 생성, __add__ 호출")
39 add = u + v
40 print("7) print add = ", add,"₩n")
41
42 print("8) 객체 sub 생성, __sub__ 호출")
43 sub = u - v
44 print("9) print sub = ", sub,"₩n")
45
46 print("10) __eq__ 호출")
47 if u == v:
48     print("11) u == v")
49 else:
50     print("12) u != v")
```

```
main) 객체 u 생성
init) self.x =  0 self.y =  1
1) print u =  (0, 1)

2) 객체 v 생성
init) self.x =  1 self.y =  0
3) print v =  (1, 0)

4) 객체 w 생성
init) self.x =  1 self.y =  1
5) print w =  (1, 1)

6) 객체 add 생성, __add__ 호출
add) self.x =  0 self.y =  1
add) other.x =  1 other.y =  0
init) self.x =  1 self.y =  1
7) print add =  (1, 1)

8) 객체 sub 생성, __sub__ 호출
sub) self.x =  0 self.y =  1
sub) other.x =  1 other.y =  0
init) self.x =  -1 self.y =  1
9) print sub =  (-1, 1)

10) __eq__ 호출
equ) self.x =  0 self.y =  1
equ) other.x =  1 other.y =  0
12) u != v
```

# Python에서의 변수의 종류

- 지역 변수 – 함수 안에서 선언되는 변수
- 전역 변수 – 함수 외부에서 선언되는 변수
- 인스턴스 변수 – Class 안에 선언된 변수, 앞에 self.가 붙는다.

# 핵심 정리

- Class는 속성과 동작으로 이루어진다. 속성은 인스턴스 변수로 표현되고 동작은 Method로 표현된다.
- 객체를 생성하려면 생성자 Method를 호출한다. 생성자 Method는 __init__() 이름의 Method이다.
- 인스턴스 변수를 정의하려면 생성자 Method 안에서 self.변수이름 과 같이 생성한다.

# Homework 1

- 다음의 vector operator overload code를 class로 구현하시오.

__mul__

__neq__

- 2023. 11. 23(목) 13:00 까지 ecampus로 upload
- 화일명

mul_이름_학번.py

neq_이름_학번.py

* 파일이 여러 개일 경우 zip으로 묶어서 제출

# Homework 2

- “5. Functions“ 숙제로 제출하였던 아래의 program들을 class를 이용하여 재작성하여 다음 2023. 11. 23(목) 13:00 까지 ecampus로 upload
- 화일명

fibo_class_이름_학번.py

tree_class_이름_학번.py

hanoi_class_이름_학번.py

day_class_이름_학번.py

* 파일이 여러 개일 경우 zip으로 묶어서 제출

# Q & A