

# AI 라이브러리 활용



## 8장 객체와 클래스

이 찬 우

# 학습 목표

- 객체지향 프로그래밍을 간단히 이해한다.
- 객체의 개념을 이해한다.
- 객체와 클래스의 관계를 이해한다.
- 객체를 활용하여 프로그램을 작성해봅니다.



# 이번 장에서 만들 프로그램

자동차 객체를 생성합니다.

자동차의 속도는 0

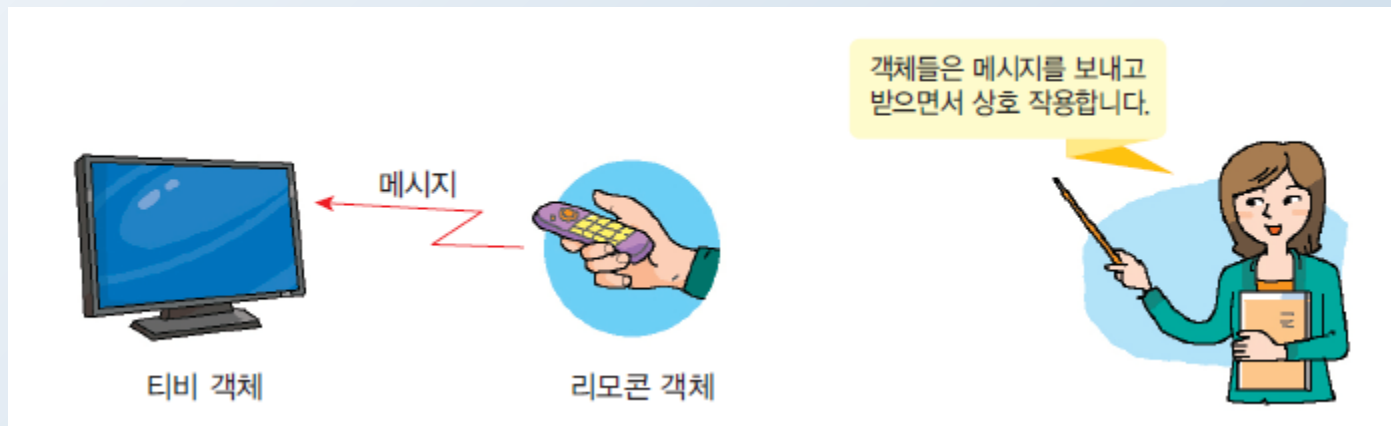
자동차의 색상은 blue

자동차의 모델은 E-class

자동차의 속도는 60

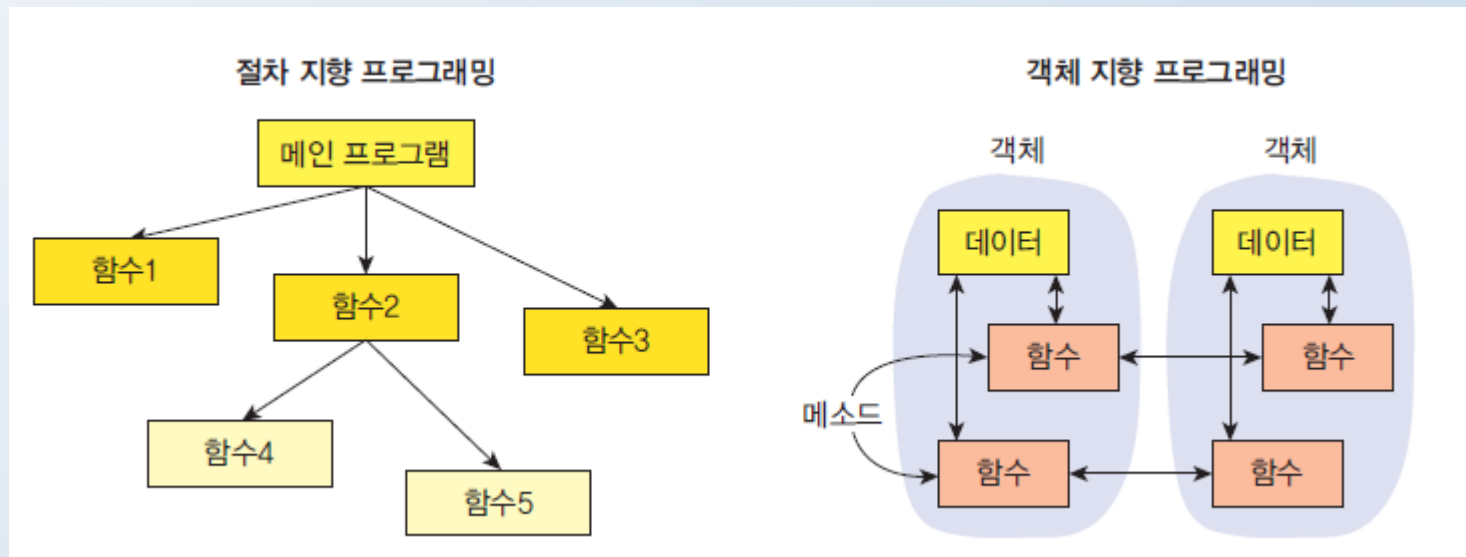
# 객체 지향 프로그래밍

- 객체 지향 프로그래밍에서는 서로 관련 있는 데이터와 함수를 묶어서 객체(object)로 만들고 이들 객체들이 모여서 하나의 프로그램이 된다.



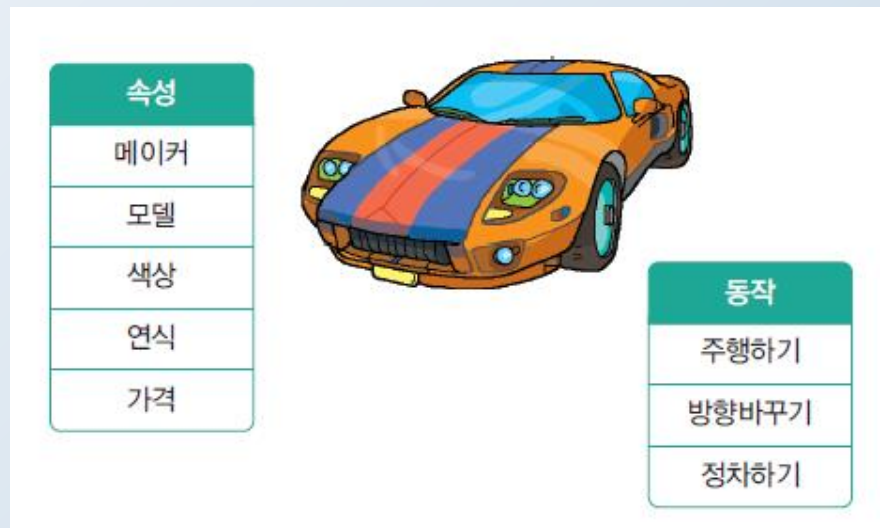
# 절차 지향과 객체 지향

- 절차 지향 프로그래밍(procedural programming)은 프로시저(procedure)를 기반으로 하는 프로그래밍 방법이다.
- 객체 지향 프로그래밍(object-oriented programming)은 데이터와 함수를 하나의 덩어리로 묶어서 생각하는 방법이다.



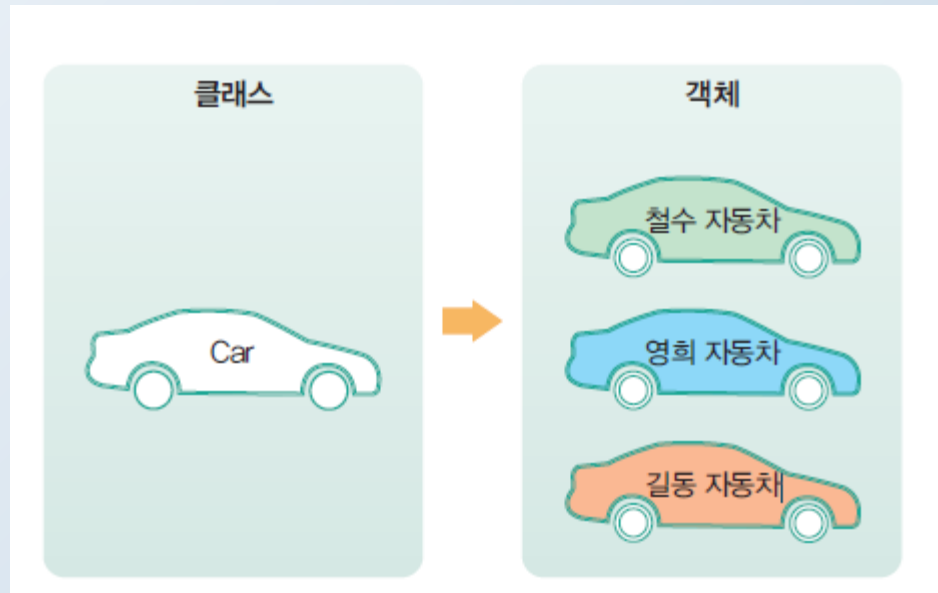
# 객체란?

- 객체(object)는 속성과 동작을 가진다.
- 자동차는 메이커나 모델, 색상, 연식, 가격 같은 속성(attribute)을 가지고 있다. 또 자동차는 주행할 수 있고, 방향을 전환하거나 정지할 수 있다. 이러한 것을 객체의 동작(action)이라고 한다.



# 클래스

- 객체에 대한 설계도를 클래스(class)라고 한다. 클래스란 특정한 종류의 객체들을 찍어내는 형틀(template) 또는 청사진(blueprint)이라고도 할 수 있다.
- 클래스로부터 만들어지는 객체를 그 클래스의 인스턴스(instance)라고 한다.



# 파이썬에서는 모든 것이 객체이다.

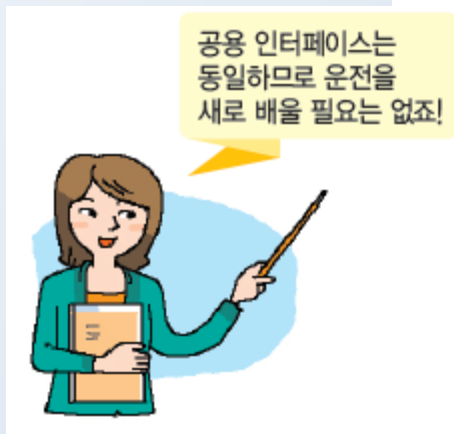
- 파이썬에서는 모든 것이 객체로 구현된다. 정수도 객체이고 문자열도 객체이며 리스트도 객체이다.

```
>>> "Everything in Python is an object".upper()  
'EVERYTHING IN PYTHON IS AN OBJECT'
```



# 캡슐화

- 공용 인터페이스만 제공하고 구현 세부 사항을 감추는 것은 **캡슐화 (encapsulation)**이라고 한다.



# Lab: TV 클래스 정의



텔레비전 객체

속성

- 채널번호
- 볼륨
- 전원상태

동작

- 켜기
- 끄기
- 채널 변경하기
- 볼륨 변경하기

객체=변수+메소드



텔레비전 객체

변수

- channel
- volume
- on

메소드

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

# 클래스 작성하기

Syntax: 클래스 정의

**형식**

```
class 클래스이름 :  
    def __init__(self, ...) :  
        ...  
    def 메소드1(self, ...) :  
        ...  
    def 메소드2(self, ...) :  
        ...
```

**예**

```
class Counter:
```

```
    def __init__(self):  
        self.count = 0
```

생성자를 정의한다.

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

메소드를 정의한다.

# Counter 클래스



```
class Counter:
```

```
def __init__(self):  
    self.count = 0
```

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

생성자 정의

인스턴스 변수 생성

# 객체 생성



```
class Counter:
```

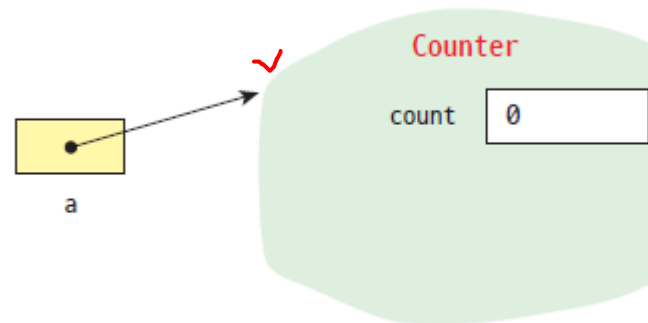
```
def __init__(self):  
    self.count = 0
```

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

생성자 정의

인스턴스 변수 생성

```
a = Counter()
```



# 객체의 멤버 접근

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

```
a = Counter()  
a.increment()  
print("카운터의 값=", a.count)
```

객체 동작

카운터의 값 = 1

# 생성자

- 생성자는 일반적으로 객체의 인스턴스 변수들을 정의하고 초기화한다.

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

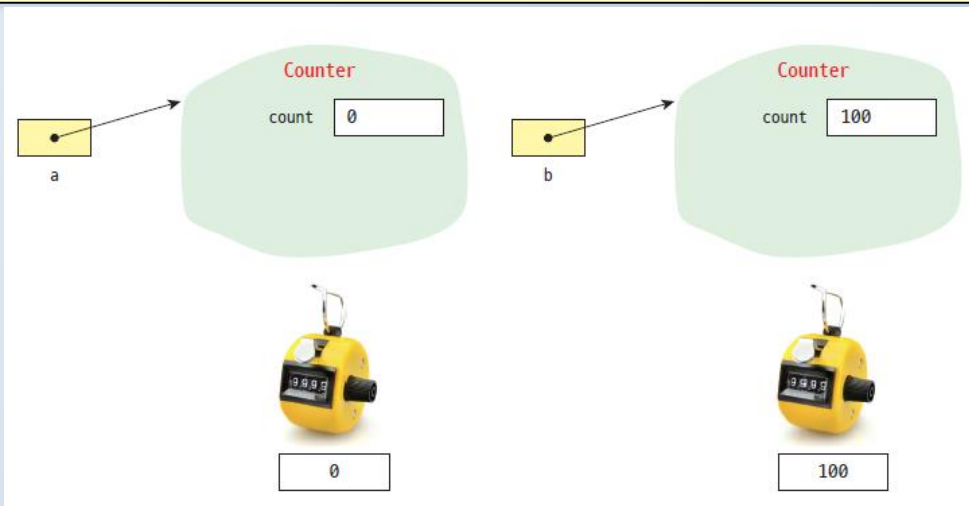
```
class Counter:
    def __init__(self, initValue=0):
        self.count = initValue
    ...
a = Counter(100) # 카운터의 초기값은 100이 된다.
b = Counter()    # 카운터의 초기값은 0이 된다.
```

생성자가 매개 변수를 가지고 있으며 만약  
사용자가 값을 전달하지 않았으면 0으로  
생각한다.

# 하나의 클래스로 객체는 많이 만들 수 있다.

```
class Counter:  
    def __init__(self, initValue=0) :  
        self.count = initValue  
  
    def increment(self) :  
        self.count += 1
```

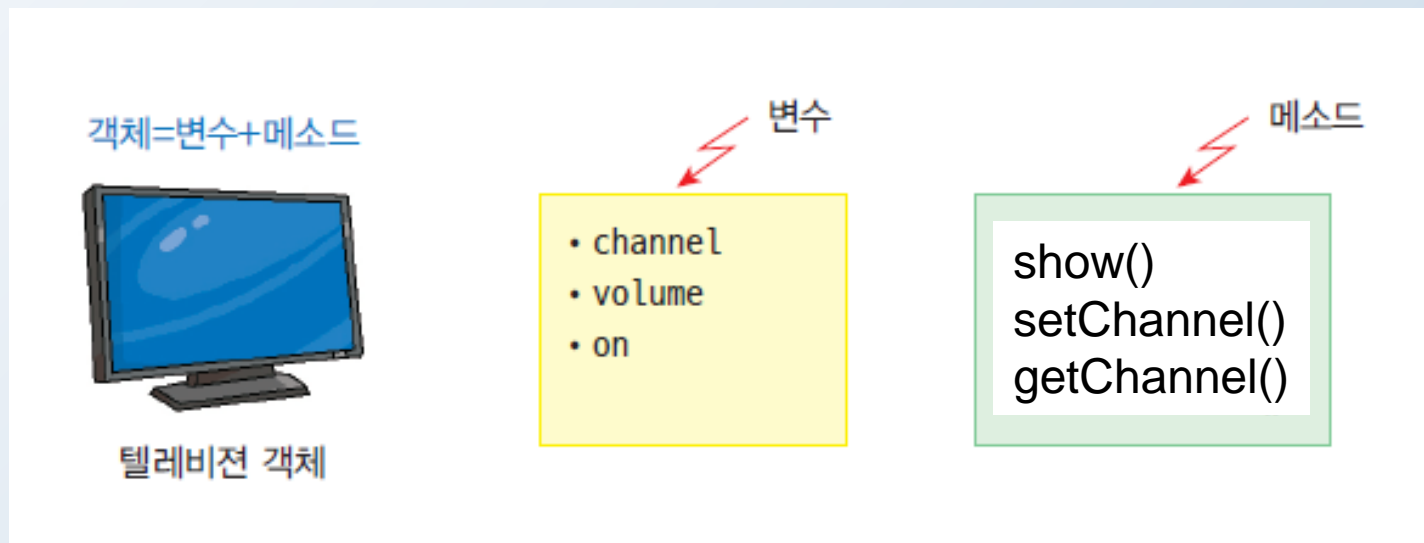
```
a = Counter(0)           # 계수기를 0으로 초기화한다.  
b = Counter(100)         # 계수기를 100으로 초기화한다.
```





# Lab:TV 클래스 정의

- TV 클래스를 작성해보자.



# Solution:

```
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
```

# Solution:

```
t = Television(9, 10, True)
```

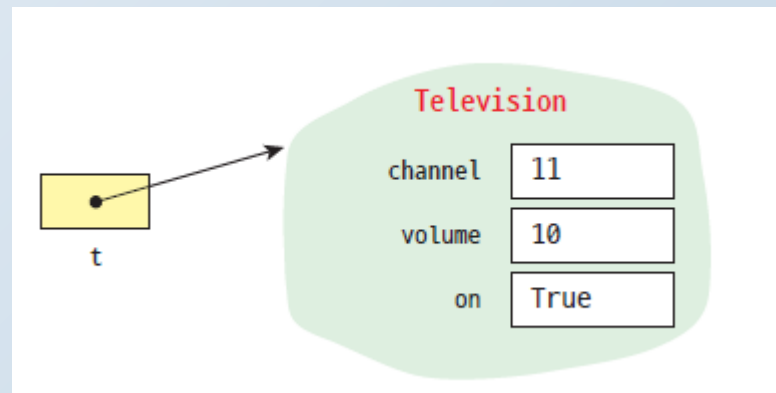
```
t.show()
```

```
t.setChannel(11)
```

```
t.show()
```

```
9 10 True
```

```
11 10 True
```



# Lab: 원 클래스 정의

- 원을 클래스로 표현해보자. 클래스 이름은 **Circle**로 하자. 원을 초기화하는 생성자는 만들어야 한다. 원은 반지름을 속성으로 가진다. 메소드로는 원의 넓이와 둘레를 반환하는 `getArea()`와 `getPerimeter()`를 정의한다.

원의 면적 314.1592653589793  
원의 둘레 62.83185307179586

# Solution:

```
import math

# Circle 클래스를 정의한다.
class Circle:
    def __init__(self, radius = 0):
        self.radius = radius

    def getArea(self):
        return math.pi * self.radius * self.radius

    def getPerimeter(self):
        return 2 * math.pi * self.radius

# Circle 객체를 생성한다.
c = Circle(10)
print("원의 면적", c.getArea())
print("원의 둘레", c.getPerimeter())
```

# Lab: Car 클래스 정의

- 자동차는 메이커나 모델, 색상, 연식, 가격 같은 속성(attribute)을 가지고 있다. 또 자동차는 주행할 수 있고, 방향을 전환하거나 주차할 수 있다. 이러한 것을 객체의 동작(action)이라고 한다.

자동차 객체를 생성하였습니다.  
자동차의 속도는 0  
자동차의 색상은 blue  
자동차의 모델은 E-class  
자동차의 속도는 60

속성
메이커
모델
색상
연식
가격



동작
주행하기
방향바꾸기
주차하기

# Solution:

```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    def drive(self):
        self.speed = 60

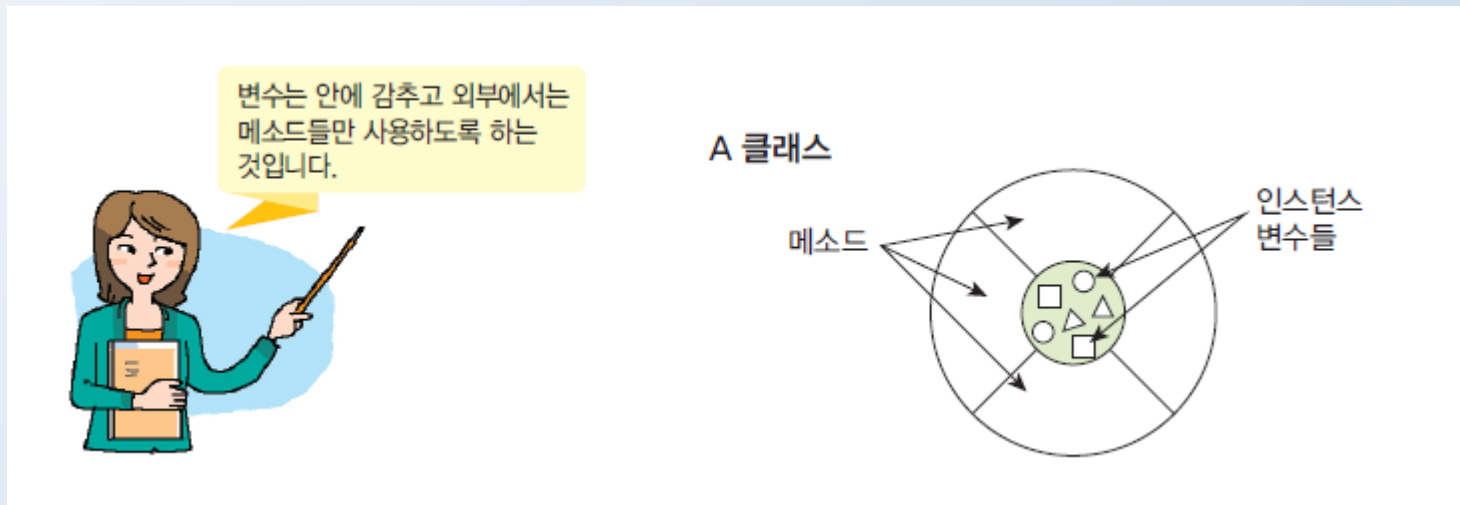
myCar = Car(0, "blue", "E-class")

print("자동차 객체를 생성하였습니다.")
print("자동차의 속도는", myCar.speed)
print("자동차의 색상은", myCar.color)
print("자동차의 모델은", myCar.model)

myCar.drive()
print("자동차의 속도는", myCar.speed)
```

# 정보 은닉

- 파이썬에서 인스턴스 변수를 **private**으로 정의하려면 변수 이름 앞에 **\_**을 붙이면 된다.
- **private**이 붙은 인스턴스 변수는 클래스 내부에서만 접근될 수 있다.





# 정보 은닉

```
class Student:
    def __init__(self, name=None, age=0):
        self.__name = name          # __가 변수 앞에 붙으면 외부에서 변경 금지
        self.__age = age            # __가 변수 앞에 붙으면 외부에서 변경 금지

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

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

# 접근자와 설정자

- 인스턴스 변수값을 반환하는 접근자(getters)
- 인스턴스 변수값을 설정하는 설정자(setters)



# 접근자와 설정자

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

    def getAge(self):
        return self.__age

    def getName(self):
        return self.__name

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

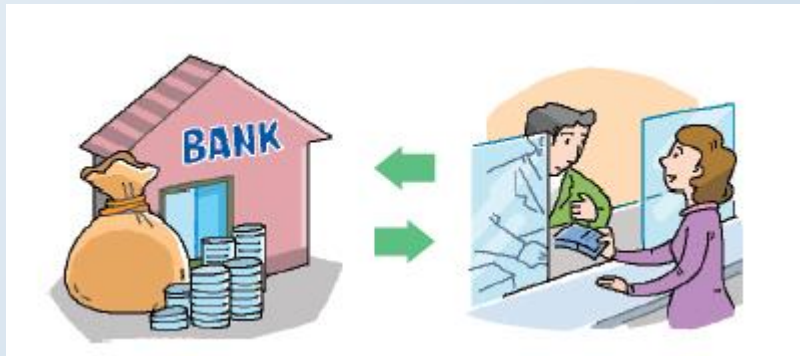
    def setName(self, name):
        self.__name=name

obj=Student("Hong", 20)
obj.getName()
```

# Lab:은행 계좌

- 우리는 은행 계좌에 돈을 저금할 수 있고 인출할 수도 있다. 은행 계좌를 클래스로 모델링하여 보자. 은행 계좌는 현재 잔액(balance)만을 인스턴스 변수로 가진다. 생성자와 인출 메소드 `withdraw()`와 저축 메소드 `deposit()` 만을 가정하자. 은행 계좌의 잔액은 외부에서 직접 접근하지 못하도록 하라.

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("통장에 ", amount, "가 입금되었음")
        return self.__balance

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

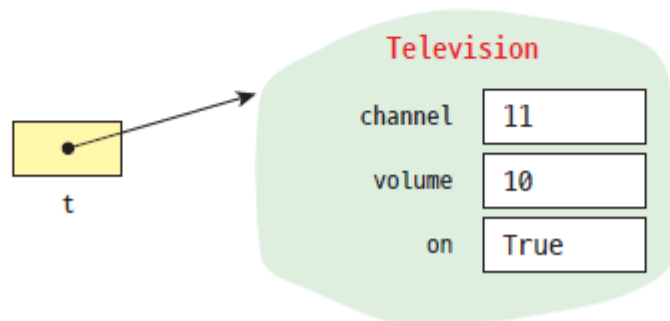
a = BankAccount()
a.deposit(100)
a.withdraw(10)
```

# 객체 참조

- 파이썬에서 변수는 실제로 객체를 저장하지 않는다. 변수는 단지 객체의 메모리 주소를 저장한다. 객체 자체는 메모리의 다른 곳에 생성된다.

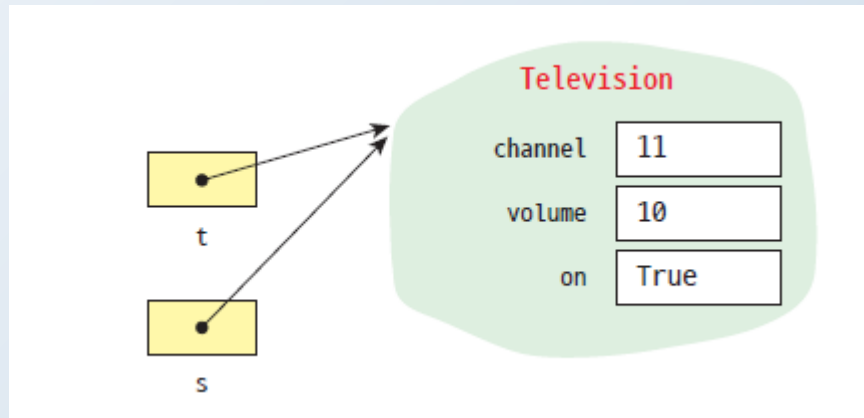
```
class Television:
    def __init__(self, channel, volume, on):
        self.channel = channel
        self.volume = volume
        self.on = on
    def setChannel(self, channel):
        self.channel = channel

t = Television(11, 10, True)
```



# 참조 공유

```
t = Television(11, 10, True)  
s = t
```

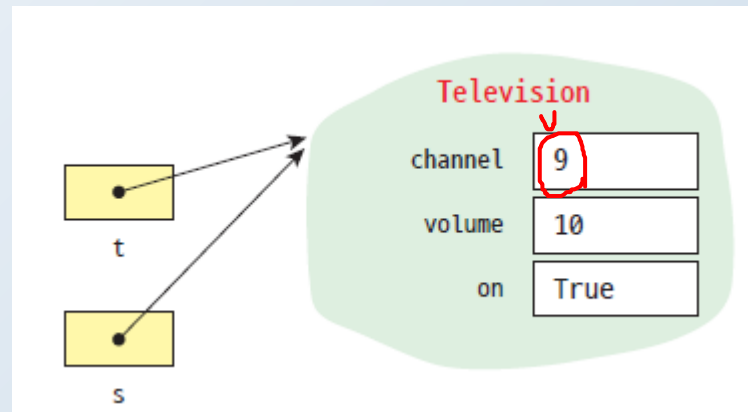


# 참조 공유

```
t = Television(11, 10, True)
```

```
s = t
```

```
s.channel = 9
```





# is, is not

- 2개의 변수가 동일한 객체를 참조하고 있는지를 검사하는 연산자가 있다. 바로 `is`와 `is not` 이다.

```
if s is t :  
    print("2개의 변수는 동일한 객체를 참조하고 있습니다.")
```

```
if s is not t :  
    print("2개의 변수는 다른 객체를 참조하고 있습니다.")
```

# None 참조값

- 변수가 현재 아무것도 가리키고 있지 않다면 **None**으로 설정하는 것이 좋다.
- **None**은 아무것도 참조하고 있지 않다는 것을 나타내는 특별한 값이다.

```
myTV = None
```

```
if myTV is None :
```

```
    print("현재 TV가 없습니다. ")
```

# 객체를 함수로 전달할 때

```
# 텔레비전을 클래스로 정의한다.
```

```
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)
```

```
# 전달받은 텔레비전의 음량을 줄인다.
```

```
def setSilentMode(t):
```

```
    t.volume = 2
```

```
# setSilentMode()을 호출하여서 객체의 내용이 변경되는지를 확인한다.
```

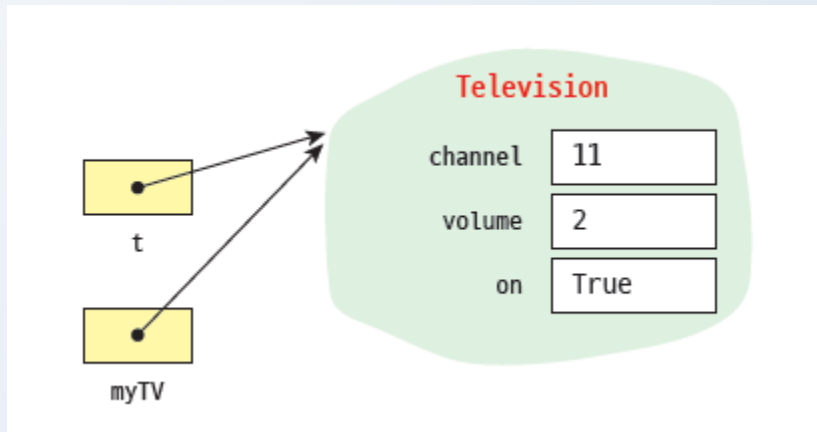
```
myTV = Television(11, 10, True);
```

```
setSilentMode(myTV)
```

```
myTV.show()
```

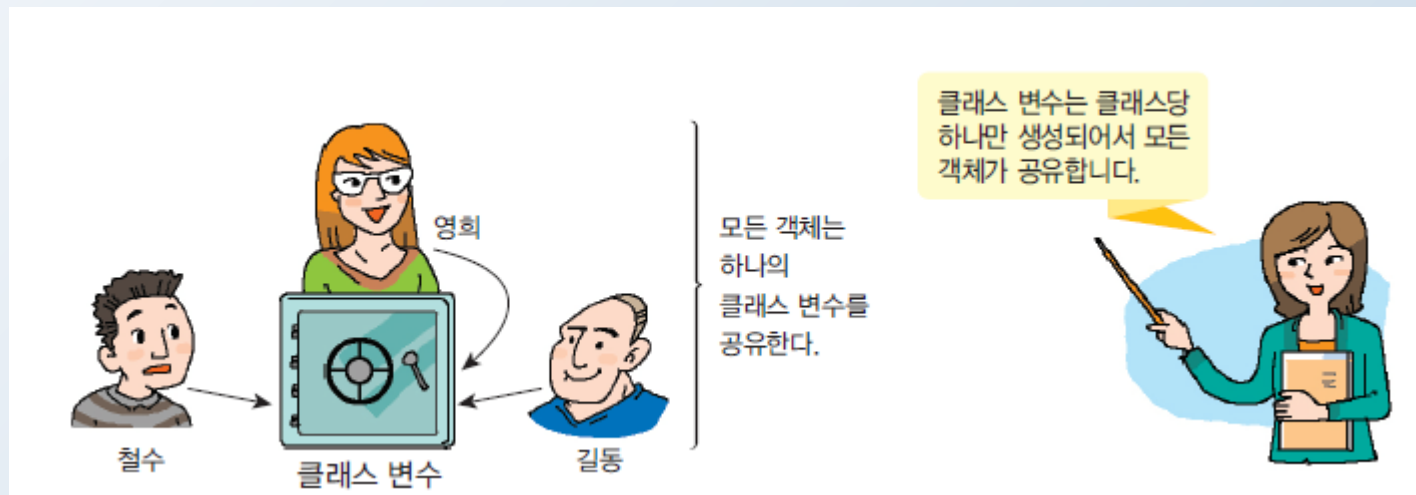
```
11 2 True
```

# 객체를 함수로 전달할 때

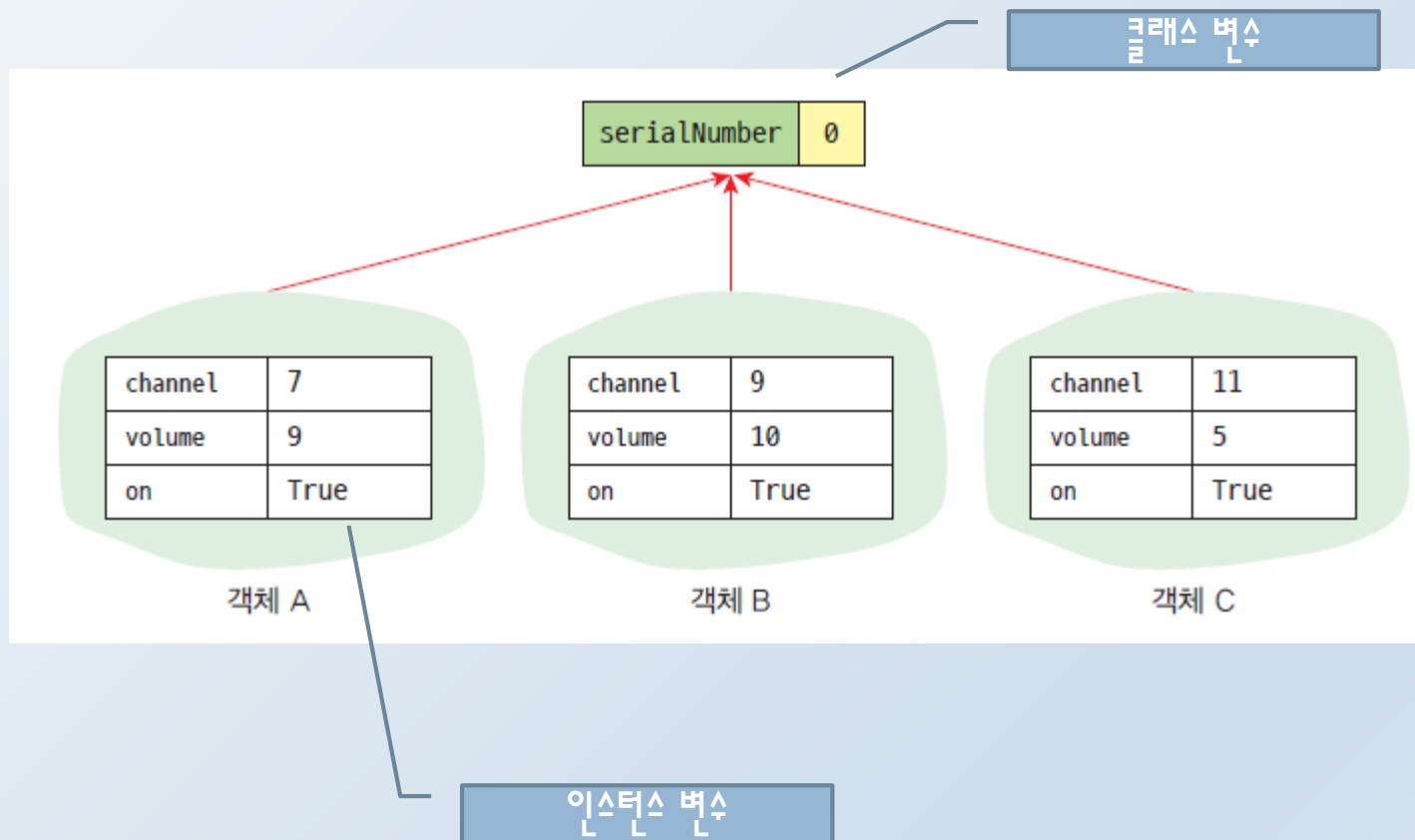


# 클래스 변수

- 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유하는 변수를 클래스 멤버(class member)라고 한다.



# 인스턴스 변수 vs 클래스 변수



# 클래스 변수

```
# 텔레비전을 클래스로 정의한다.
class Television:
    serialNumber = 0                # 이것이 클래스 변수이다.

    def __init__(self, channel, volume, on):
        self.channel = channel
        self.volume = volume
        self.on = on
        Television.serialNumber += 1    # 클래스 변수를 하나 증가한다.
        # 클래스 변수의 값을 객체의 시리얼 번호로 한다.
        self.number = Television.serialNumber

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

myTV = Television(11, 10, True);
myTV.show()
```

11 10 True 1

# 상수 정의

- 상수들은 흔히 클래스 변수로 정의된다.

```
class Monster :  
    # 상수 값 정의  
    WEAK = 0  
    NORMAL = 10  
    STRONG = 20  
    VERY STRONG = 30  
  
    def __init__(self) :  
        self._health = Monster.NORMAL  
  
    def eat(self) :  
        self._health = Monster.STRONG  
  
    def attack(self) :  
        self._health = Monster.WEAK
```



# Lab: 클래스 변수

- 어떤 섬에 강아지들이 있는데 강아지의 품종은 모두 같다고 하자. 그렇다면 강아지 객체마다 품종을 저장할 필요는 없을 것이다. 강아지의 품종은 클래스 변수로 정의하여도 된다.

```
class Dog:
    kind = "Bulldog"          # 클래스 변수
    def __init__(self, name, age):
        self.name = name      # 각 인스턴스에 유일한 인스턴스 변수
        self.age = age        # 각 인스턴스에 유일한 인스턴스 변수
```

# 특수 메소드

- 파이썬에는 연산자(+, -, \*, /)에 관련된 특수 메소드(special method)가 있다.
- 이들 메소드는 우리가 객체에 대하여 +, -, \*, /와 같은 연산을 적용하면 자동으로 호출된다.

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

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

# 특수 메소드

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

# Lab:은행 계좌

- 2차원 공간에서 벡터(vector)는  $(a, b)$ 와 같이 2개의 실수로 표현될 수 있다. 벡터 간에는 덧셈이나 뺄셈이 정의된다.

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

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

- 특수 메소드를 이용하여서  $+$  연산과  $-$  연산, `str()` 메소드를 구현해보자.

$$(0, 1) + (1, 0) = (1, 1)$$

```
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)
```

# Lab: 주사위 클래스

- 주사위의 속성
  - ▣ 주사위의 값(value)
  - ▣ 주사위의 위치(x, y)
  - ▣ 주사위의 크기(size)
- 주사위의 동작
  - ▣ 주사위를 생성하는 연산(`__init__`)
  - ▣ 주사위를 던지는 연산(`roll_dice`)
  - ▣ 주사위의 값을 읽는 연산(`read_dice`)
  - ▣ 주사위를 화면에 출력하는 연산(**`print_dice`**)



```
from random import randint
```

```
class Dice :
```

```
    def __init__(self, x, y) :
```

```
        self._x = x
```

```
        self._y = y
```

```
        self._size = 30
```

```
        self._value = 1
```

```
    def read_dice(self) :
```

```
        return self._value
```

```
    def print_dice(self) :
```

```
        print("주사위의 값=", self._value)
```

```
    def roll_dice(self) :
```

```
        self._value = randint(1, 6)
```

```
d = Dice(100, 100)
```

```
d.roll_dice()
```

```
d.print_dice()
```

# 이번 장에서 배운 것

- 클래스는 속성과 동작으로 이루어진다.  
속성은 인스턴스 변수로 표현되고 동작은 메소드로 표현된다.
- 객체를 생성하려면 생성자 메소드를 호출한다.  
생성자 메소드는 `__init__()` 이름의 메소드이다.
- 인스턴스 변수를 정의하려면 생성자 메소드 안에서 `self`.  
변수이름 과 같이 생성한다.





# 다음 시간 수업

딥러닝에서 데이터 다루기

