

## 8장 객체와 클래스

# 학습 목표

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



# 이번 장에서 만들 프로그램

자동차 객체를 생성하겠습니다.

자동차의 속도는 0

자동차의 색상은 blue

자동차의 모델은 E-class

자동차의 속도는 60

# 객체 지향 프로그래밍

- 서로 관련 있는 데이터와 함수를 묶어서 객체(object)로 만들고 이들 객체들이 모여서 하나의 프로그램이 된다.
- 객체지향 프로그래밍(OOP: object-oriented programming)은 우리가 사는 실제 세계가 객체들로 구성된 것과 비슷하게, 소프트웨어도 객체로 구성하는 방법이다.
- 실제 세계에는 사람, 자동차, 텔레비전, 세탁기, 냉장고 등의 많은 객체가 존재한다. 객체들은 객체 나름대로 고유한 기능을 수행하면서 다른 객체들과 메시지를 통하여 상호 작용한다.



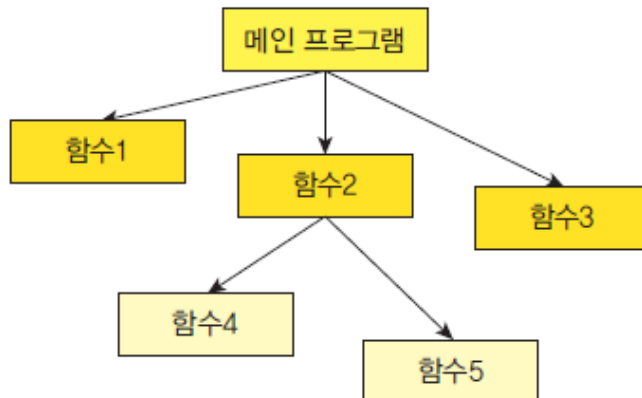
객체들은 메시지를 보내고  
받으면서 상호 작용합니다.



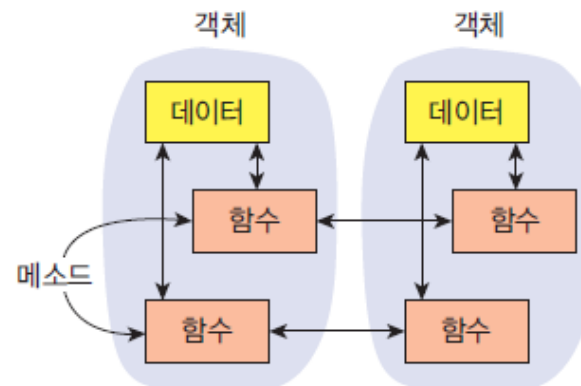
# 절차 지향과 객체 지향

- 절차 지향 프로그래밍(procedural programming) : 프로시저 (procedure = 함수)를 기반으로 하는 프로그래밍. 서로 관련된 데이터와 함수를 묶을 수 없다. 함수 작성에만 신경. 예-자동차에서 속도를 나타내는 변수 **spped**와 속도를 변경하는 함수 **accel()**은 하나로 묶여 있는 것이 합리적이지만 절차지향에서는 불가능
- 객체 지향 프로그래밍(object-oriented programming) : 데이터와 함수를 하나의 덩어리로 묶어서(캡슐화) 생각하는 방법. 현재 가장 각광 받는 기술.

절차 지향 프로그래밍



객체 지향 프로그래밍



# 객체란

- 객체(object)는 하나의 물건이라 생각하면 된다.
- 속성과 동작을 가진다.

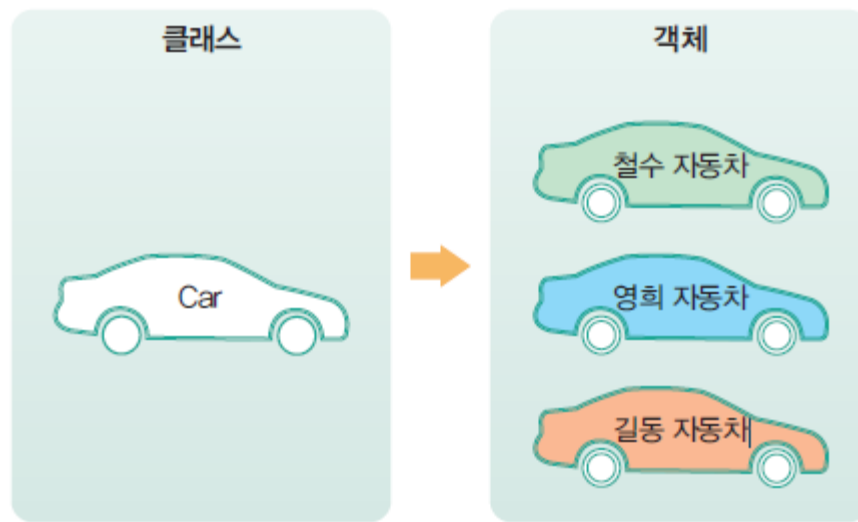
속성
메이커
모델
색상
연식
가격



동작
주행하기
방향바꾸기
정차하기

# 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.
- 객체는 우리가 사용할 수 있는 메소드를 가지고 있다. 예를 들어서 리스트는 `insert()`나 `remove()`와 같은 메소드를 가지고 있다. 문자열은 `upper()`와 같은 메소드를 가지고 있다.

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



# 캡슐화 비결

- 공용 인터페이스만 제공하고 구현 세부 사항을 감추는 것은 **캡슐화(encapsulation)**이라고 한다.
- 어떻게 동작되는지는 알 필요 없고 사용하는 방법만 알면 된다



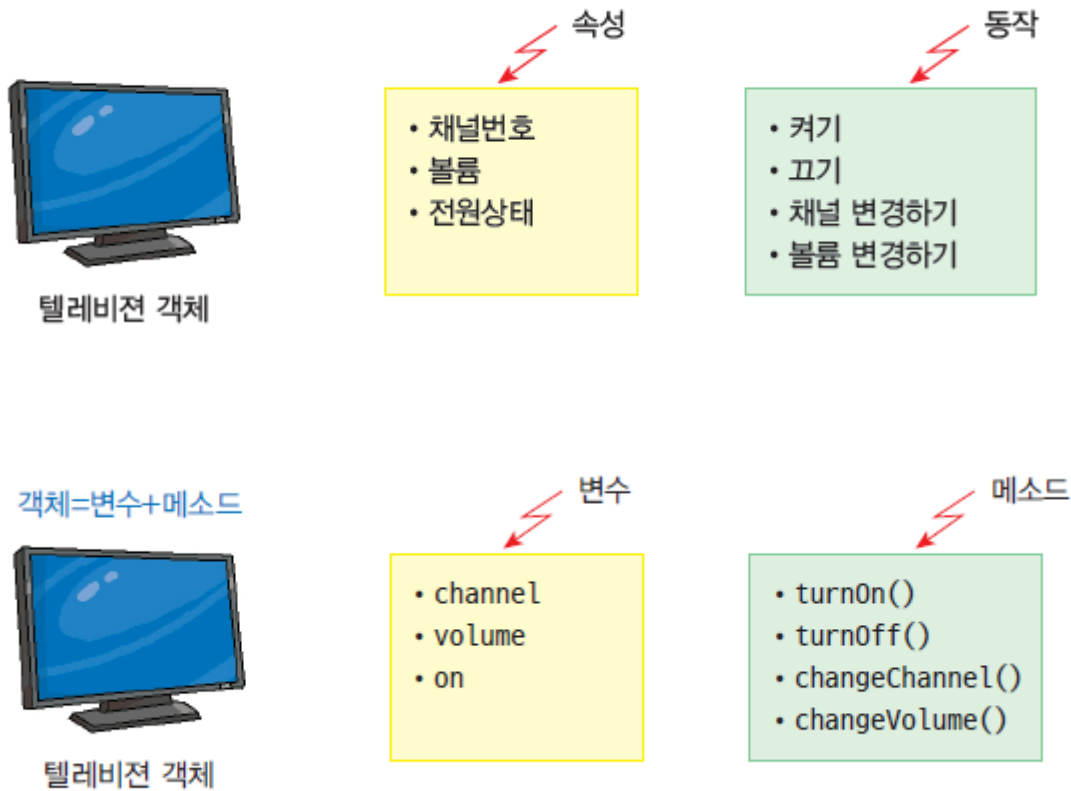
# 정거장

1. 객체 지향 프로그래밍은 들을 조합하여서 프로그램을 작성하는 기법이다.
2. 절차 지향 프로그래밍의 가장 큰 단점은 무엇인가?
3. 캡슐화를 다시 설명해보자.
4. 클래스는 무엇인가?
5. 인스턴스는 무엇인가?
6. 클래스와 인스턴스의 관계는 무엇인가?



# Lab: TV 클래스 정의

- TV를 나타내는 클래스를 정의해보자. 어떤 속성과 동작이 TV 클래스에 있을까?



# 클래스 작성하기

- 클래스는 객체의 형태를 정의하는 틀(template)과 같은 것이다. 객체를 찍어내는 틀이라고 생각해도 된다.

Syntax: 클래스 정의

**형식**

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

**예**

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

생성자를 정의한다.

메소드를 정의한다.

# Counter 클래스

- 예: 경기장이나 콘서트에 입장하는 관객 수를 세는 계수기를 Counter 클래스로 만든다면.



```
class Counter:
```

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

```
        self.count = 0
```

```
    def increment(self):
```

```
        self.count += 1
```

생성자 정의

인스턴스 변수 생성

- 클래스 이름의 첫 글자는 일반적으로 대문자로
- `__init__()` 는 생성자(constructor). 객체를 초기화하는 메소드. 클래스로 객체를 생성할 때 디폴트로 호출된다.
- 자기자신을 가르키는 **self** 사용

# 객체 생성



```
class Counter:
```

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

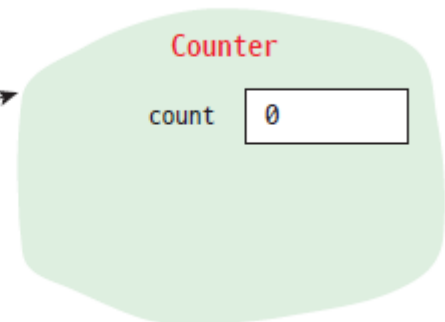
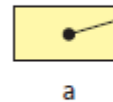
생성자 정의

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

인스턴스 변수 생성

- 클래스 이름에 ()를 붙여 함수처럼 호출하면 객체가 생성된다. 객체가 생성되면서 생성자 메소드인 `__init__()`가 자동으로 호출된다.

```
a = Counter()
```



# 객체의 멤버 접근

- 객체 이름에 점(.)을 붙여 메소드 이름 또는 변수 이름을 적어준다.

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

p374.py

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

객체.동작

카운터의 값= 1

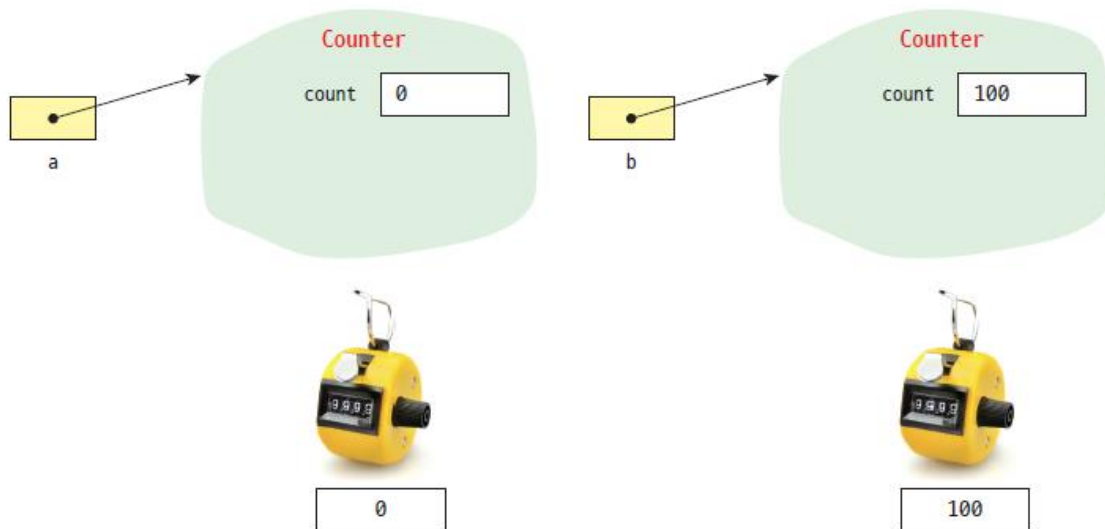
# 하나의 클래스로 개체는 많이 만들 수 있다.

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

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

p375.py

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





# 하나의 클래스로 개체는 많이 만들 수 있다.

## □ 변수의 종류

지역변수 – 함수 안에서 선언되는 변수

전역변수 – 함수 외부에서 선언되는 변수

인스턴스 변수 – 클래스 안에서 선언된 변수, 앞에 **self.**가 붙는다.

인스턴스 변수의 범위는 클래스 전체가 된다.

지역변수는 메소드 안에서 생성한 변수로 범위는 메소드 안이 된다.

소스 안에서 **self**를 붙이지 않으면 지역변수가 된다.

```
def show():
```

```
    s = “현재 설정값”    #지역변수
```

```
    print(s, self.count)
```

# 장점

1. 클래스를 정의하는 데 사용되는 구문은 무엇인가?
2. 클래스 이름은 일반적으로 어떻게 작명하는가?
3. 클래스의 인스턴스를 어떻게 생성하는가?
- 4 인스턴스의 속성과 메소드에 어떻게 접근하는가?
5. 메소드는 무엇인가?
6. `self`의 목적은 무엇인가?
7. `__init__()` 메소드의 목적은 무엇인가?



# Lab: TV 클래스 정의

- TV 클래스를 작성해보자.

```
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)

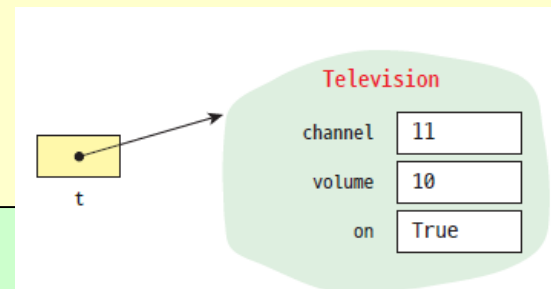
...
...
```

p377.py

```
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.

```
import math
```

p378.py

```
# 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
```

```
...
```

원의 면적 314.1592653589793

원의 둘레 62.83185307179586

# Lab: 자동차 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.

```
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)
```

```
...
...
```

p379.py

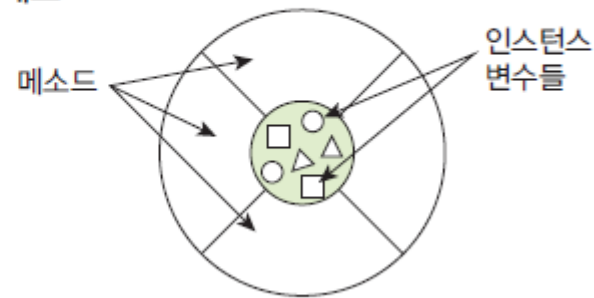


# 정보 인니

- 클래스 안의 데이터를 외부에서 마음대로 바꾸지 못하게 하는 것
- 인스턴스 변수를 **private**으로 정의. 변수 이름 앞에 **\_**을 붙이면 된다.
- **private**이 붙은 인스턴스 변수는 클래스 내부에서만 접근될 수 있다.



A 클래스



# 정보 인니

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

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

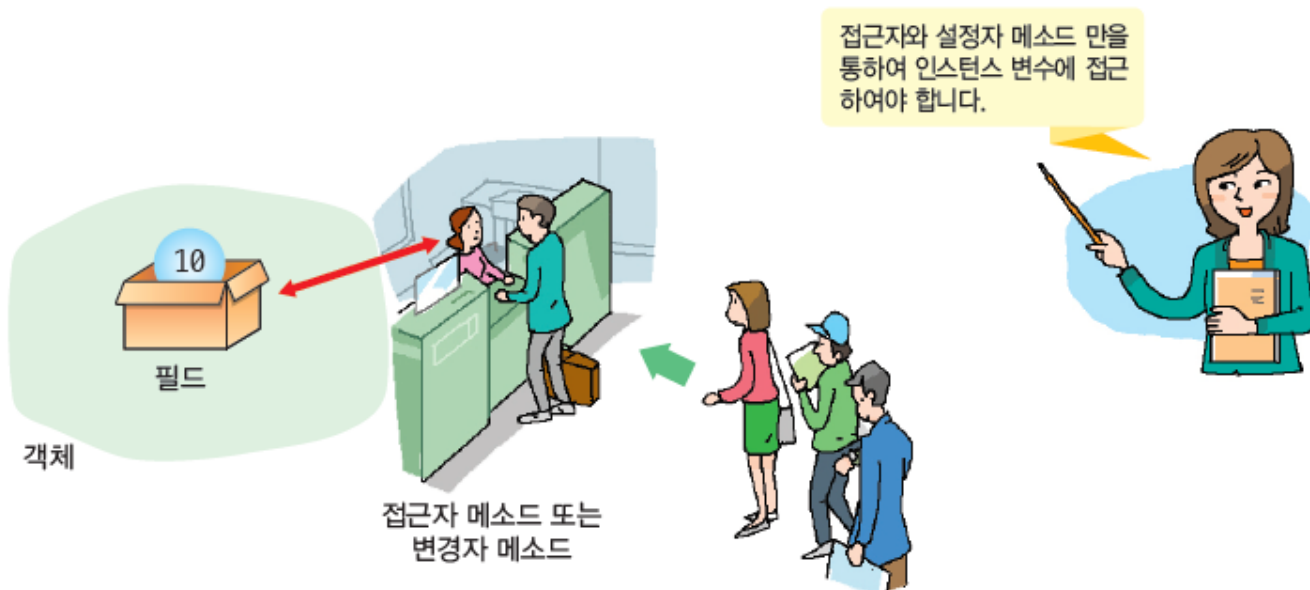
p381.py

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

- 클래스 멤버에 대한 접근을 제어하는 것은 객체 지향 프로그래밍의 핵심적인 부분이다. 접근을 제어하게 되면 객체를 잘못 사용하는 것을 방지할 수 있다.
- 메소드에 대해서도 똑같이 할 수 있다. 메소드를 정의할 때에도 앞에 `__`를 붙일 수 있고 그 의미는 변수의 경우와 동일하다.

# 접근자와 설정자

- `private`으로 된 인스턴스 변수와 메소드를 클래스 외부에서 사용할 수 있게 하는 방법
- 접근자(**getters**) : 인스턴스 변수값을 반환
- 설정자(**setters**) : 인스턴스 변수값을 설정





# 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()
```

p383.py

인스턴스 변수가 private로 정의되어  
이더라도 외부에서는 접근자와 설정자  
메소드를 이용하여 불변이 인스턴스  
변수의 값을 변경하거나 읽을 수 있다.

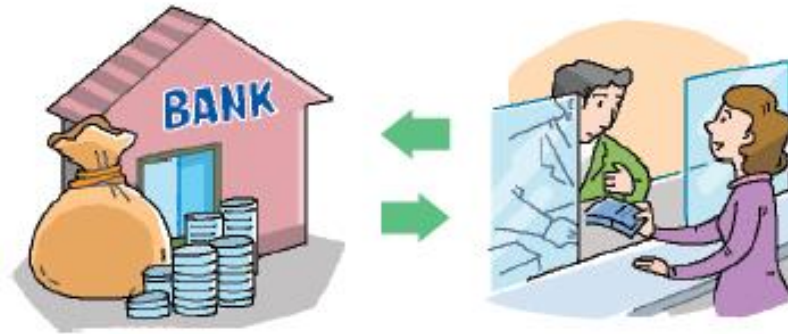
# 장점

1. 정보는닉은 왜 필요한가?
2. 접근자와 설정자의 이름은 어떻게 정하는가?
3. 클래스 안에서 어떤 변수를 **private**로 만들려면 어떻게 하면 되는가?



# 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.



# 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)
```

p384.py

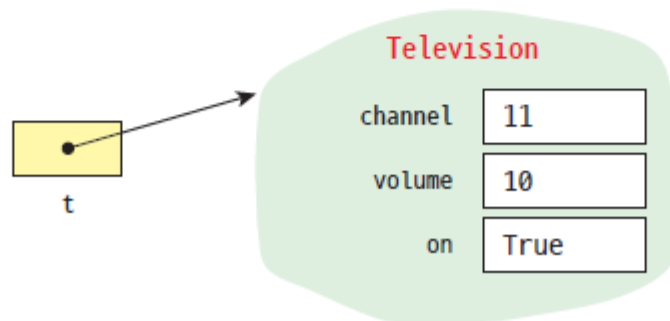
```
통장에서 100 가 출금되었음
통장에 10 가 입금되었음
```

# 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

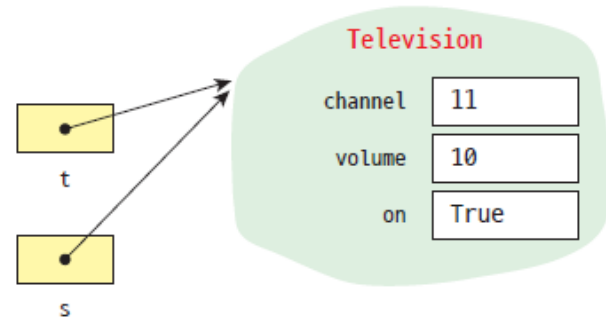
t = Television(11, 10, True)
```



# 참조 공유

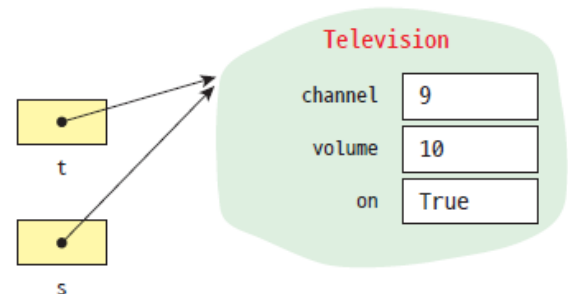
- 객체를 복사하면 객체가 복사되는 것이 아니라 참조값(객체의 주소)만 복사되어서 변수 **s**에 저장된다.

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



- **s**를 통해 객체를 수정하면 **t**가 가르키는 객체의 값도 변경된다.

```
t = Television(11, 10, True)
s = t
s.channel = 9
```



# 참조 공유

- is와 is not : 2개의 변수가 동일한 객체를 참조하고 있는지를 검사하는 연산자

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

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

# None 참조가

- **None** : 변수가 현재 아무것도 가리키고 있지 않다. 아무것도 참조하고 있지 않다.는 의미

```
myTV = None
```

```
if myTV is None :
```

```
    print("현재 TV가 없습니다. ")
```



# 객체를 함수로 전달할 때

- 객체가 함수의 매개변수로 전달되어 함수 안에서 변경할 수 있다.

# 텔레비전을 클래스로 정의한다.

p387.py

```
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):
```

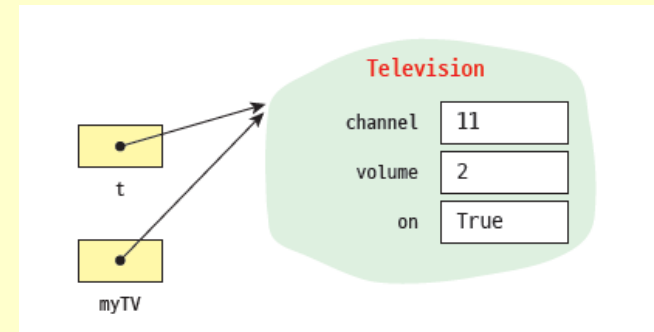
```
    t.volume = 2
```

# setSilentMode()을 호출하여서 객체의 내용이 변경되는지를 확인한다.

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

```
setSilentMode(myTV)
```

```
myTV.show()
```



11 2 True

# 정가정거

1. 객체를 함수로 전달하면 원본이 전달되는가? 아니면 복사본이 전달되는가?

2 다음과 같은 문장이 실행되면 내부적으로 어떤 일이 일어나는가?

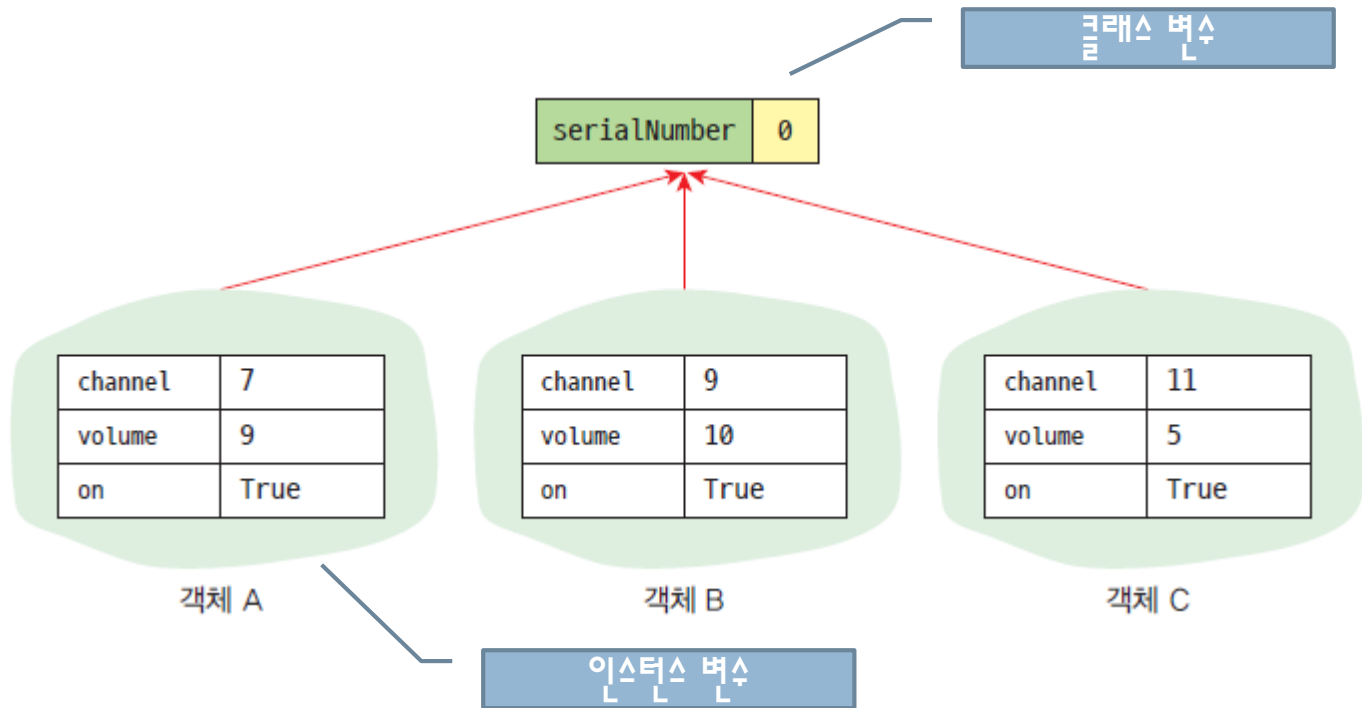
```
t = Television(11, 10, True)
```

```
s = t
```



# 인스턴스 변수 vs 클래스 변수

- 인스턴스 변수 - 객체가 생성될 때 별도로 생성되는 변수
- 클래스 변수 - 모든 객체가 공유하는 변수. 하나의 클래스에 하나만 존재. 메서드 외부에 변수를 생성



# 인스턴스 변수 vs 클래스 변수

# 텔레비전을 클래스로 정의한다.

p390py

```
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

11 10 True 2

# 상수 정의

- 상수들은 흔히 클래스 변수로 정의해서 사용한다.

```
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
```

# 정가점거 중간점검

1. 인스턴스 변수와 클래스 변수의 차이점은 무엇인가?
2. 파이썬에서 클래스 변수를 생성하려면 어떻게 하면 되는가?



# 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"          # 모든 인스턴스 변수들이 공유하는 클래스 변수
    def __init__(self, name, age=0):
        self.name = name      # 각 인스턴스에 유일한 인스턴스 변수
        self.age = age        # 각 인스턴스에 유일한 인스턴스 변수

a = Dog("Baduk", 2)
b = Dog("Marry", 3)

print(a.kind)                 # 모든 강아지가 공유
print(b.kind)                 # 모든 강아지가 공유
print(Dog.kind)               # 모든 강아지가 공유
...
...
```

p392.py

# 특수 메소드

- 연산자(+, -, \*, /)에 관련된 특수 메소드(special method)가 있다.
- 이들 메소드는 우리가 객체에 대하여 +, -, \*, /와 같은 연산을 적용하면 자동으로 호출된다. 특수 메서드를 이용하면 객체의 상황에 맞는 자연스러운 연산을 정의할 수 있다.

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

# 특수 메소드

- 예. `__str__()` 메서드 : `print()`로 출력할 때 자동적으로 호출되는 특수 메소드

```
class Counter:                                     p393.py
    def __init__(self, count) :
        self.count = count
    def increment(self):
        self.count += 1
    def __str__(self):
        msg = "카운트값:" + str(self.count)
        return msg

a = Counter(100)
print(a)
```

```
카운트값:100
```

# Lab: 벡터 객체에 특수 메소드 정의하기

- 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)

...
...
```

p394.py

# 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
```

p396.py

```
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.

