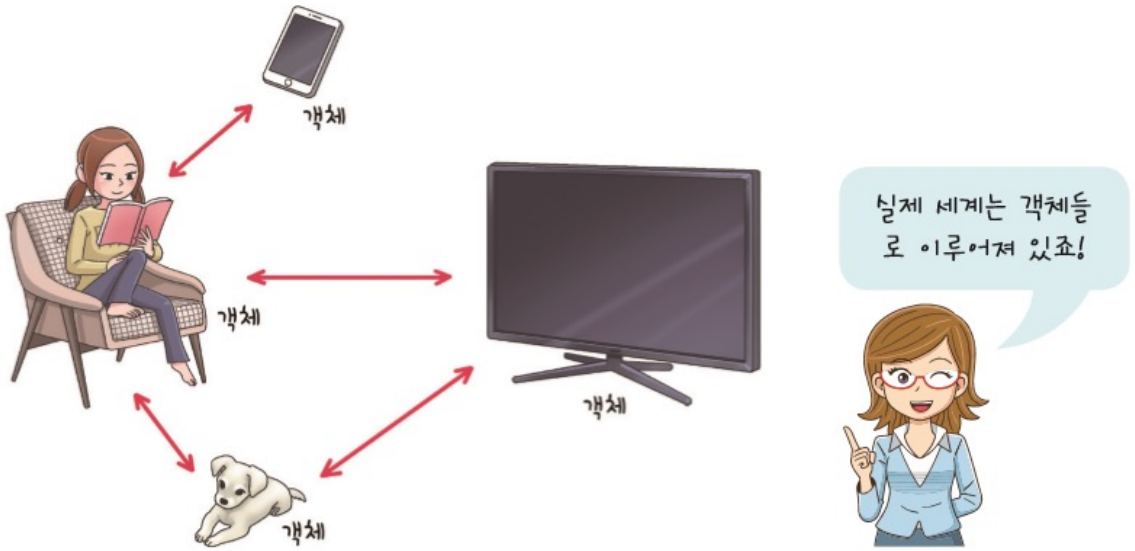

# 파이썬 프로그래밍

## 9장. 클래스와 객체

# 객체지향 프로그래밍

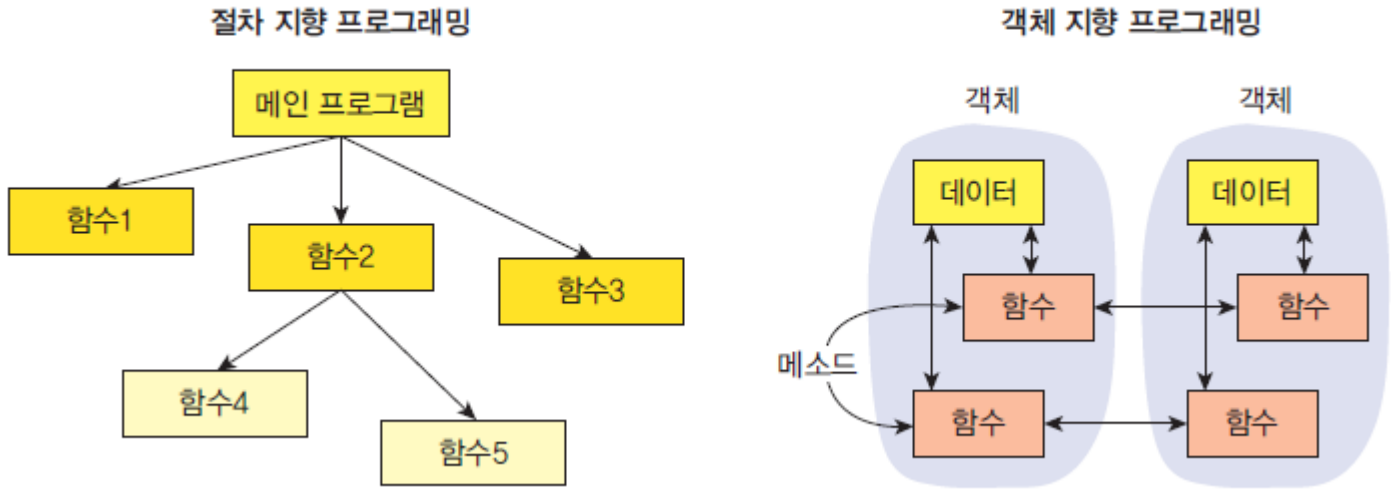
- **객체 지향 프로그래밍(OOP: object-oriented programming)**
  - 우리가 살고 있는 실제 세계가 객체(object)들로 구성되어 있음
    - 실제 세계에는 사람, 텔레비전, 세탁기, 냉장고 등 많은 객체가 존재함
  - 소프트웨어를 객체로 구성하는 방법
    - 객체들은 고유한 기능을 수행하며, 다른 객체들과 상호 작용을 함



# 절차 지향과 객체 지향

- 절차 지향 프로그래밍(procedural programming)
  - 프로시저(procedure)를 기반으로 하는 프로그래밍 방법

- 객체 지향 프로그래밍(object-oriented programming)
  - 데이터와 함수를 하나의 덩어리로 묶어서 객체(object)만듦
  - 이들 객체들이 모여서 하나의 프로그램을 구성하는 방법



# 객체란?

- 객체는 속성(attribute)과 동작(action)을 가짐

- 객체의 예: 자동차
  - 속성: 메이커, 모델, 색상, 연식, 가격
  - 동작: 주행, 방향 전환, 정차 등

| 속성 |
| --- |
| 메이커 |
| 모델 |
| 색상 |
| 연식 |
| 가격 |

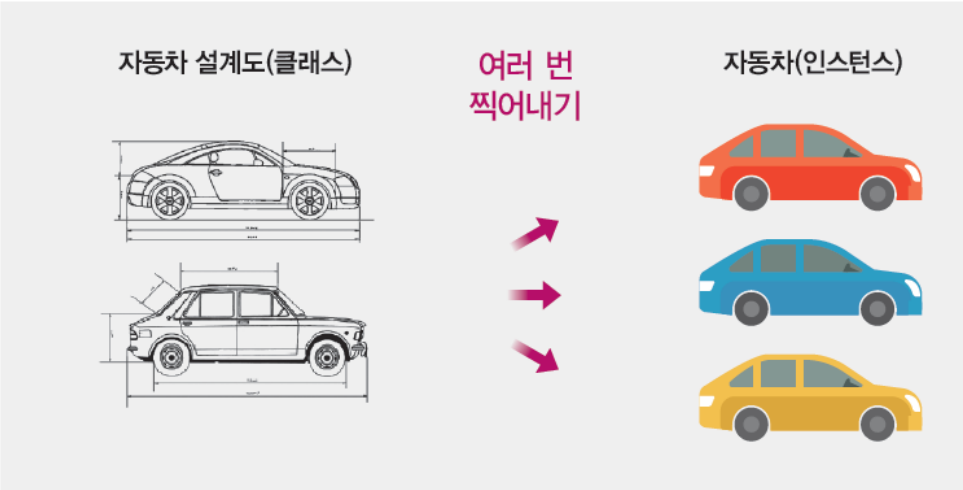
| 동작 |
| --- |
| 주행하기 |
| 방향바꾸기 |
| 정차하기 |

# 클래스란?

- 클래스(class)
  - 객체에 대한 설계도

- 인스턴스(instance)
  - 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance)라고 함

- 파이썬에서는 모든 것이 객체로 구성
  - 정수, 문자열, 리스트도 객체임
  - 객체의 특징: 사용할 수 있는 메소드를 가지고 있음

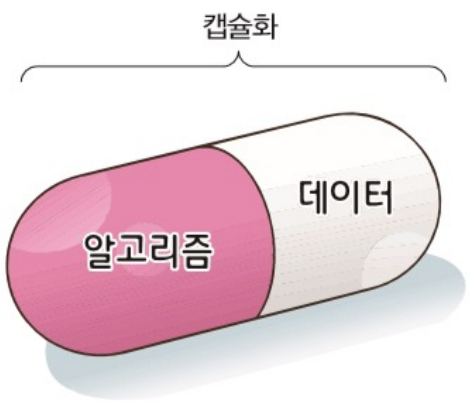




설계도 완성품

# 캡슐화

- 캡슐화(encapsulation)
  - 공용 인터페이스만 제공하고 구현 세부 사항을 감추는 것

- 정보 은닉: 공용 인터페이스만 제공함





# 클래스 작성하기

- 모든 메소드의 매개변수에 self가 추가됨
- 클래스 멤버
  - 인스턴스 변수
  - 메소드: 객체(object)와 연관되어 있는 함수
    - str.split(), str.append()

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 작성

- Counter 클래스 작성
  - Counter 클래스는 기계식 계수기를 나타내며 경기장이나 콘서트에 입장하는 관객 수를 세기 위하여 사용할 수 있다.

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

생성자 정의

인스턴스 변수 생성

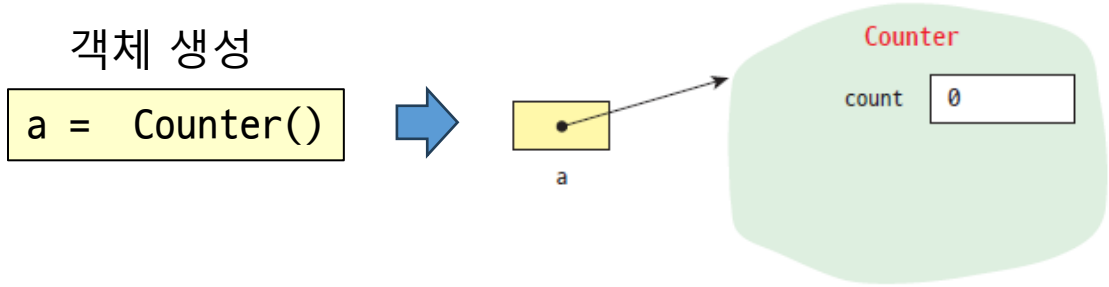
# 객체 생성

- 객체 생성: 클래스이름()
  - 함수처럼 호출하면 객체가 생성됨: a = Counter()
  - 객체가 생성되면서 생성자 메소드인 __init__()가 자동으로 호출

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

생성자 정의

인스턴스 변수 생성




Counter 클래스 객체

# 객체의 멤버 접근

- 멤버 접근
  - 객체 이름에 점(.)을 붙임
  - 객체 메소드 호출: **객체이름.함수(인수)**
    - a.increment()
  - 객체 멤버 변수 접근: **객체이름.변수**
    - a.count

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


a = Counter() # Counter 클래스의 객체 생성
a.increment()
print("카운터 a의 값은", a.count)
```

객체의 멤버 변수 접근

```
카운터 a의 값은 1
```

# Counter 클래스 및 객체 생성

- 인스턴스 변수
  - 생성자 내부에서 초기화
    - self.count = 0
  - 변수의 범위
    - 클래스 전체에서 사용 가능
  - 항상 self가 붙음

자동 호출

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

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

    def reset(self):
        self.count = 0

    def get(self):
        return self.count


a = Counter() # Counter 클래스의 객체 생성
print(a.count)
a.increment()
print("카운터 a의 값은", a.get())
print("카운터 a의 값은", a.count)
```

인스턴스 변수 선언 및 초기화

인스턴스 변수는 클래스 멤버 함수에서 사용 가능

```
0
카운터 a의 값은 1
카운터 a의 값은 1
```

# 생성자

- 생성자(constructor)
  - 객체가 생성될 때 객체의 인스턴스 변수들을 정의하고 초기화함
  - 파이썬은 하나의 클래스에 하나의 생성자만 허용
  - 객체가 생성될 때 자동으로 호출됨

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

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

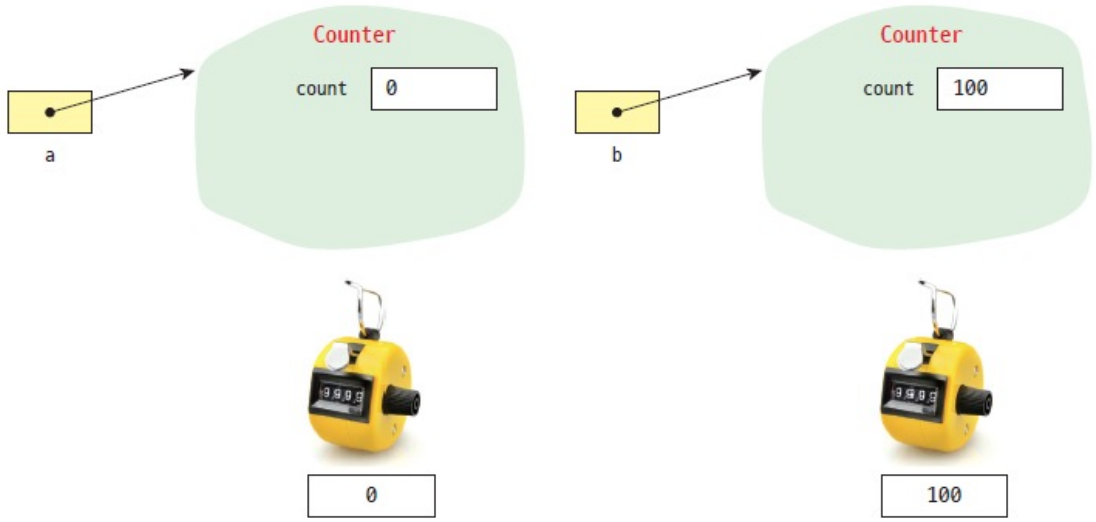
```
class Counter:
    def __init__(self, init_value=0):
        self.count = init_value

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

a = Counter(100) # 카운터의 초기값은 100
b = Counter()    # 카운터의 초기값은 0
```

생성자에 매개변수 정의

# 하나의 클래스로 많은 객체 생성

- 여러 객체 생성

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

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

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



# 인스턴스 변수와 지역 변수

- 인스턴스 변수
  - 생성자 내부에서 생성
  - self.변수명
  - 인스턴스 변수 범위: 클래스 전체
- 지역 변수
  - 함수(메소드)안에서 생성된 변수
  - self를 붙이지 않음
  - 지역 변수의 범위: 메소드 내부
- self 매개 변수
  - 객체 자신을 참조하는 변수
    - self.변수이름
    - self.메소드이름()
  - 객체 a와 b를 구분
    - 어떤 객체가 show()함수를 호출했는지 구분
    - show(a), show(b)라고 호출하지 않음
  - a.show()를 호출
    - show()메소드의 self 매개변수에 a가 전달됨

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

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

    def show(self):
        s = "현재 설정값"
        print(s, self.count) # self.count: 인스턴스 변수

a = Counter(0)
a.show()

b = Counter(100)
b.show()
```

클래스 내부의 메소드를 호출할 때 self.메소드이름() 사용

지역 변수 s
- 메소드 내부에서 선언
- 메소드 내부에서만 사용 가능

```
현재 설정값 1
현재 설정값 101
```

# Lab. 자동차 클래스(Car) 정의

- 자동차 클래스 속성
  - 속성: 속도, 색상, 모델
  - 동작: 주행

| 속성 |
| --- |
| 메이커 |
| 모델 |
| 색상 |
| 연식 |
| 가격 |

| 동작 |
| --- |
| 주행하기 |
| 방향바꾸기 |
| 주차하기 |

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

# Solution. 자동차 클래스(Car) 정의

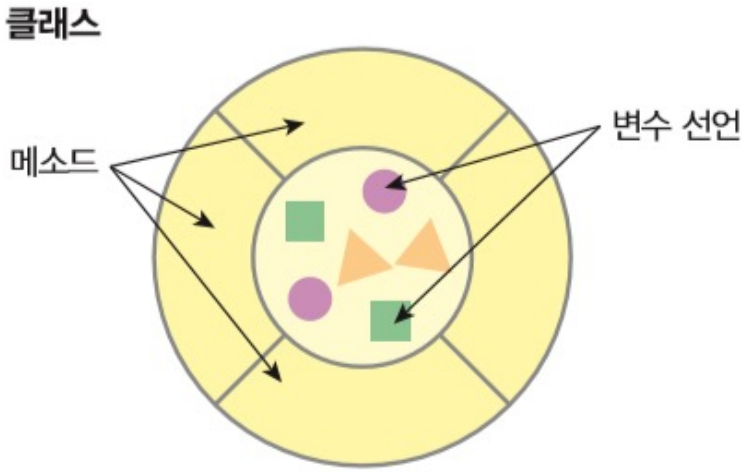
```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    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(Information hiding)

- 구현의 세부 사항을 클래스 안에 감추는 것
  - 클래스 안의 데이터를 외부에서 마음대로 변경하지 못하게 하는 것

- 인스턴스 변수를 private 으로 정의
  - 변수 이름 앞에 __을 붙이면 된다.

- private이 붙은 인스턴스 변수
  - 클래스 내부에서만 접근될 수 있음




# 정보 은닉 예제

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

    def print_info(self):
        print(f'name: {self.__name}, age: {self.__age}')


obj=Student('HongGilDong', 21)
obj.print_info()
print(obj.__age)
```

private 변수 선언

private 변수 사용
- 클래스 내부에서만 사용 가능

private 변수 __age를 클래스 외부에서 접근: AttributeError

```
name: Hong, age: 21
Traceback (most recent call last):
  File "/Users/changsu/workspace_swcoding/ch09_class/student.py", line 12, in <module>
    print(obj.__age)
AttributeError: 'Student' object has no attribute '__age'
```

# 접근자와 설정자

- 접근자와 설정자 사용 이유
  - 정보 은닉과 연관: private 변수에 접근하기 위한 방법

- 접근자(getter) 메소드
  - 인스턴스 변수값을 반환하는 메소드
  - getXXX() 이름 사용

- 설정자(setter) 메소드
  - 인스턴스 변수값을 설정(변경)하는 메소드
  - setXXX() 이름 사용

# 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("Alice", 20)
print(obj.getName())

obj.setName("Charlie")
print(obj.getName())
```

접근자: private 변수 접근

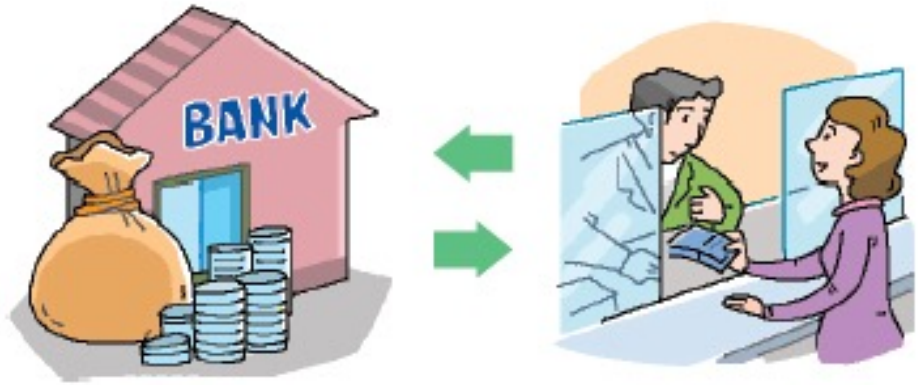
설정자: private 변수 변경

객체의 private 변수 접근 및 변경

```
Alice
Charlie
```

# Lab: 은행 계좌

- 은행 계좌를 클래스로 모델링 하기
  - 은행 계좌는 현재 잔액(balance)만을 인스턴스 변수로 가짐
  - 생성자와 인출 메소드 withdraw()와 저축 메소드 deposit() 만을 가정
  - 은행 계좌의 잔액은 외부에서 직접 접근하지 설정할 것



# Solution: 은행 계좌

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

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

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



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

```
통장에서  100 원 입금되었음
통장에서  10 원 출금되었음
```

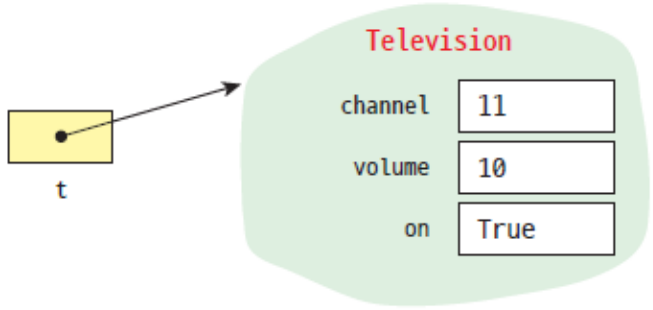
# 객체 참조 (Object Reference)

- 객체 변수는 실제로 객체를 저장하지 않음
  - 객체의 메모리 주소를 저장
  - 객체의 주소: 객체 참조값(object reference)

- 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)
```

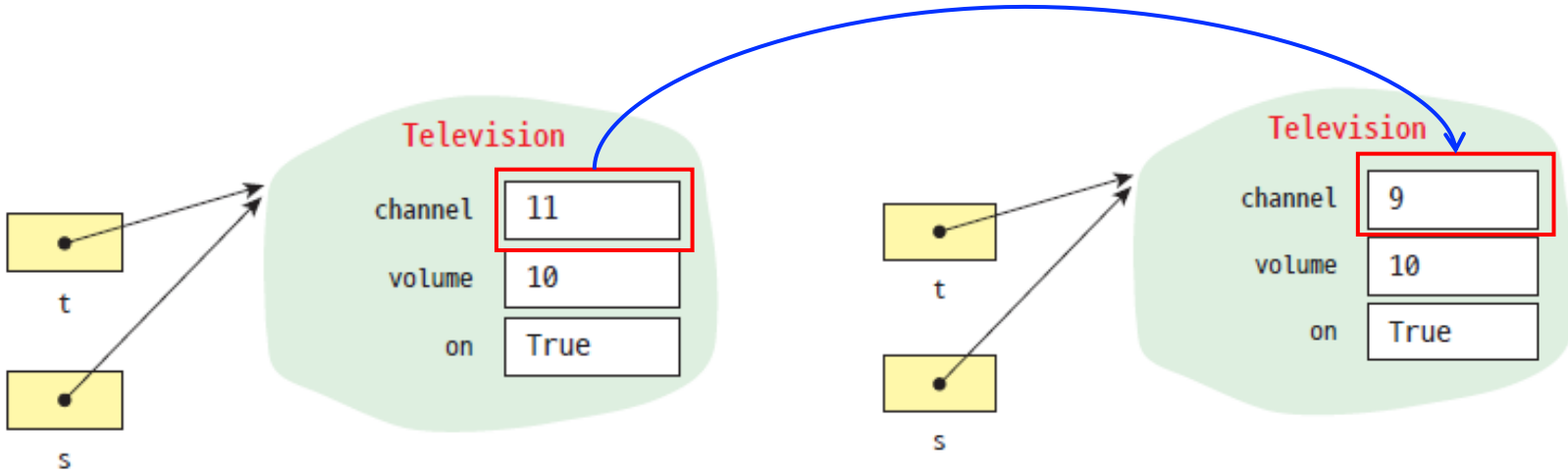
변수 t는 객체의 주소를 저장



# 참조 공유

- 참조 공유
  - 객체의 참조값을 저장하고 있는 변수를 다른 변수로 복사
  - 객체가 복사되는 것이 아니라 객체의 주소만 복사되어 저장
  - 동일한 객체를 가리키게 됨

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



# is, is not

- is, is not 연산자
  - 2개의 변수가 동일한 객체를 참조하고 있는지 검사하는 연산자

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

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

    def get_channel(self):
        return self.channel

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

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

if s is t:
    print("2개의 변수는 동일한 객체를 참조")

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

```
9 10 True
9 10 True
2개의 변수는 동일한 객체를 참조
```

# None 참조값

- 변수가 아무것도 가리키고 있지 않는 경우,
  - None으로 설정: 아무것도 참조하고 있지 않는다는 의미

```
my_tv = None

if my_tv is None:
    print("현재 TV가 없습니다.")
```

# 객체를 함수로 전달

- 객체가 함수의 파라미터로 전달되고 객체를 변경한 경우
  - 변경 불가능한 객체
    - 함수 안에서 변경된 내용이 적용 안됨
  - 직접 작성한 객체
    - 함수가 객체의 내용을 변경

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

    def show(self):
        print(f'channel:{self.channel}, '
              f'volume:{self.volume}, on:{self.on}')
```

```
def set_silent_mode(t):
    t.volume = 2

# set_silent_mode()를 호출하여
# 객체의 내용이 변경되는지 확인

my_tv = Television(11, 10, True)
my_tv.show()

set_silent_mode(my_tv)
my_tv.show()
```

Television 객체를 파라미터로 받음

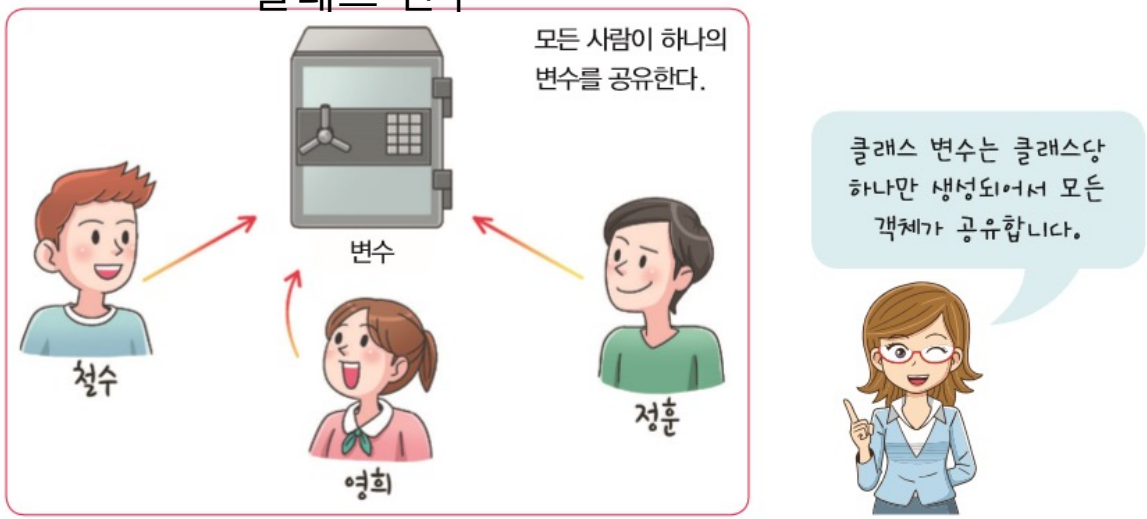
```
channel:11, volume:10, on:True
channel:11, volume:2, on:True
```

# 클래스 변수

- 객체 변수(인스턴스 변수)
  - 항상 객체를 통해서 생성되고 사용됨
  - 객체마다 독립된 값을 가짐

- 클래스 변수
  - 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유
  - 이러한 변수를 정적 멤버 또는 클래스 변수라고 한다.

클래스 변수



# 인스턴스 변수 vs 클래스 변수

- 인스턴스(객체) 변수
  - 객체(인스턴스)가 생성될 때마다 생성
  - 각 객체는 이들 변수에 별도의 기억 공간을 가지고 있음

| channel | 7 |
|---|---|
| volume | 9 |
| on | True |

Television객체 A

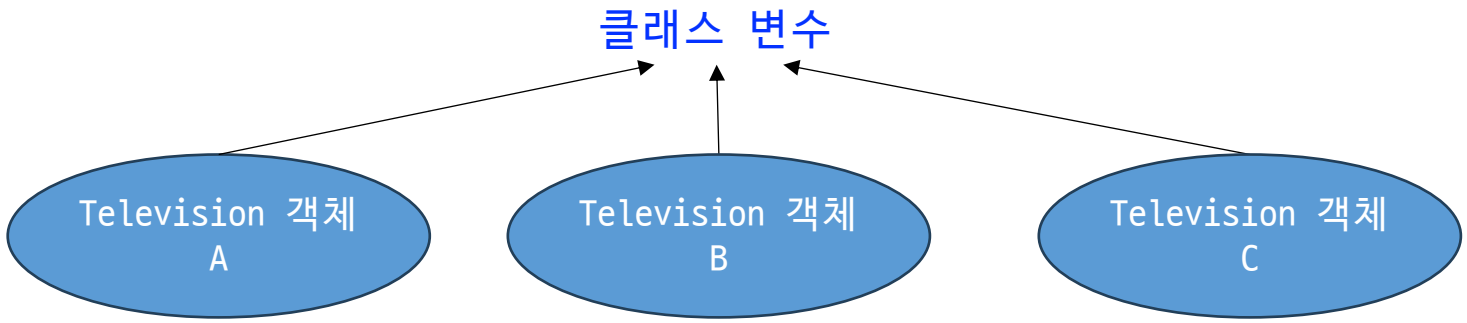
| channel | 9 |
|---|---|
| volume | 10 |
| on | True |

Television객체 B

| channel | 11 |
|---|---|
| volume | 5 |
| on | False |

Television객체 C

- 클래스 변수
  - 객체(인스턴스)가 공유하는 변수
  - 클래스명.변수명으로 접근

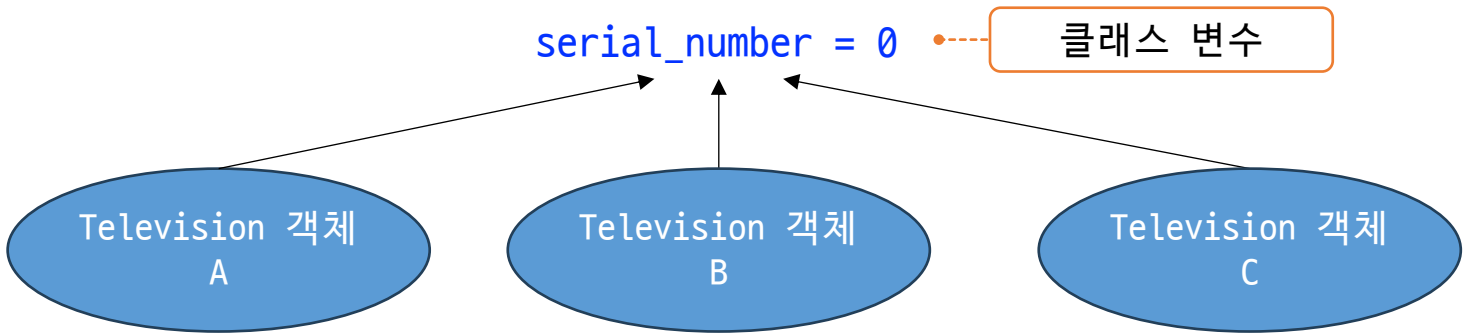


# 클래스 변수

- 클래스 변수
  - 선언: 생성자 이전에 선언
  - 클래스명.변수명으로 접근: Television.serial_number

```
class Television:
    serial_number = 0

    def __init__(self, channel, volume, on):
        Television.serial_number += 1
        self.number = Television.serial_number # 클래스 변수 사용
        self.channel = channel
        self.volume = volume
        self.on = on
```

클래스 변수
- 생성자 이전에 선언



# 클래스 변수 예제

```
class Television:
    serial_number = 0

    def __init__(self, channel, volume, on):
        Television.serial_number += 1
        self.number = Television.serial_number
        self.channel = channel
        self.volume = volume
        self.on = on

    def show(self):
        print(f'channel:{self.channel}, '
              f'volume:{self.volume}, on:{self.on}')

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

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

생성자에서 serial_number의 값을 1씩 증가

```
t1 = Television(7, 9, True)
t2 = Television(9, 10, True)
t3 = Television(11, 5, False)

t1.show()
t2.show()
t3.show()
print()

print(f't1.serial_number:{t1.serial_number}, id:{id(t1.serial_number)}')
print(f't2.serial_number:{t2.serial_number}, id:{id(t2.serial_number)}')
print(f't3.serial_number:{t3.serial_number}, id:{id(t3.serial_number)}')
print(f'Television.serial_number:{Television.serial_number}, '
      f'id:{id(Television.serial_number)}')
```

```
channel:7, volume:9, on:True
channel:9, volume:10, on:True
channel:11, volume:5, on:False

t1.serial_number:3, id:4339902832
t2.serial_number:3, id:4339902832
t3.serial_number:3, id:4339902832
Television.serial_number:3, id:4339902832
```

각 객체의 serial_number 변수의 id는 동일함

# Lab: 객체 생성과 사용

- 상자를 나타내는 Box 클래스를 작성하여 보자. Box 클래스는 가로 길이, 세로 길이, 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)


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

__str__(self) 메소드
- 객체 자체를 문자열 형태로 출력할 때 형식을 지정해 주는 함수

print(객체)
  __str__(self) 메소드가 자동 호출

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

# 특수 메소드

- 특수 메소드
  - 파이썬에는 연산자(+, -, *, /)와 관련된 **특수 메소드(special method)**가 존재
  - 이들 메소드는 객체에 대하여 +, -, *, / 연산을 적용하면 자동 호출됨

```
class Circle:
    def __init__(self, radius):
        self.radius = radius

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

c1 = Circle(10)
c2 = Circle(20)

if c1 == c2:
    print("두 원의 반지름은 동일합니다. ")
else:
    print("두 원의 반지름은 다릅니다. ")
```

__eq__(self, other) 자동 호출

두 원의 반지름은 다릅니다.

# 특수 메소드

| 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 |

# 특수 메소드 예제: 벡터

- 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)
```

# 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)
print(w-u)
```

__add__(self, other)
자동 호출

```
(1, 1)
(1, 0)
```

# 주사위 클래스

- 주사위의 속성
  - 주사위의 값(value)
  - 주사위의 면의 수(faceNum)

- 주사위의 동작
  - 주사위를 생성하는 연산: `__init__(faceNum)`
  - 주사위를 던지는 연산: `roll_dice()`
  - 주사위의 값을 읽는 연산: `read_dice()`
  - 주사위를 화면에 출력하는 연산: `print_dice()`

# 주사위 클래스 예제

```
from random import randint

class Dice:
    def __init__(self, face_number):
        self.__facenum = face_number  # 주사위 면의 수
        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, self.__facenum)


face_num = int(input("주사위의 면의 수를 입력하세요: "))
d = Dice(face_num)

count = int((input("주사위를 던질 회수를 입력하세요: ")))
for i in range(count):
    d.roll_dice()
    d.print_dice()
```

```
주사위의 면의 수를 입력하세요: 6
주사위를 던질 회수를 입력하세요: 10
주사위의 값= 2
주사위의 값= 1
주사위의 값= 1
주사위의 값= 4
주사위의 값= 3
주사위의 값= 1
주사위의 값= 5
주사위의 값= 3
주사위의 값= 3
주사위의 값= 6
```

# __repr__()과 __str__() 비교

- 두 함수 모두 객체를 문자열로 변환
  - __repr__()
    - 클래스를 문자열로 표현하는 방식
    - 공식적인 문자열 표현(official)
    - 파이썬 인터프리터에서 확인
  - __str__()
    - 클래스를 문자열로 표현하는 리턴
    - 클래스의 비공식적 문자열 표현
    - print(객체)에서 사용
  - 클래스 내부에 __str__()이 없으면
    - __repr__()이 호출됨

**repr()과 str() 비교**

```
class Card:
    def __init__(self, card_suit, card_number):
        self.suit = card_suit
        self.number = card_number

    def __repr__(self):
        '''
        객체를 공식적인 문자열로 변환(인터프리터에서 객체 표현에 사용)
        '''
        return f'[{self.suit},{self.number:>2}]'

    def __str__(self):
        '''
        객체를 문자열로 변환: print(객체)에 사용
        :return:
        '''
        return f'({self.suit},{self.number:>2})'
```

```
card = Card('♠', 10)
card  # 파이썬 인터프리터에서 객체 출력: __repr__()
```

```
[♠,10]
```

```
print(card) # print(객체)는 __str__() 호출
```

```
(♠,10)
```

# 핵심 정리

- 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.

# Questions?