12장 클래스와 객체

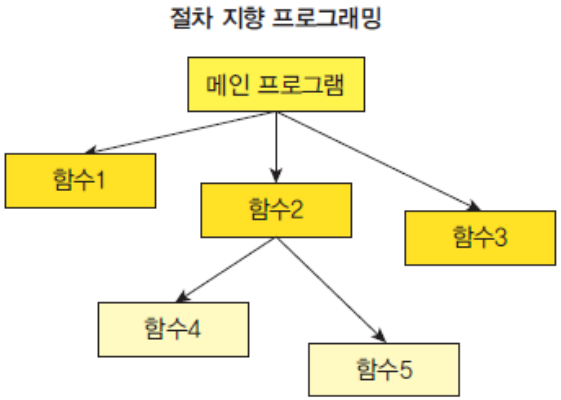
객체 지향 프로그래밍. p417

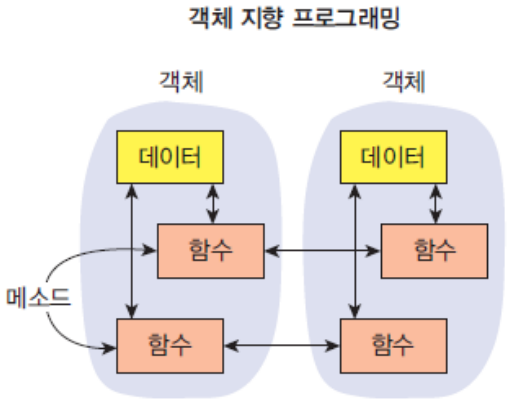
- 객체 지향 프로그래밍(OOP: object-oriented programming) : 우리가 사는 실제 세계가 객체(object)들로 구성된 것과 비슷하게, 소프트웨어도 객체로 구성하는 방법
- 실제 세계에는 사람, 자동차, 텔레비전, 세탁기, 냉장고 등의 많은 객체가 존재한다. 객체들은 객체 나름대로 고유한 기능을 수행하면서 다른 객체들과 메시지를 통하여 상호 작용한다. TV와 리모콘은 메시지를 통하여 서로 상호 작용한다.

- 소프트웨어 개발도 이와 같이 하는 방식을 객체 지향이라고 한다. 다양한 기능을 하는 소프트웨어 객체들을 작성하고, 이러한 객체들을 조합하여 자기가 원하는 기능을 구현하는 기법이다.

절차 지향과 객체 지향. p417

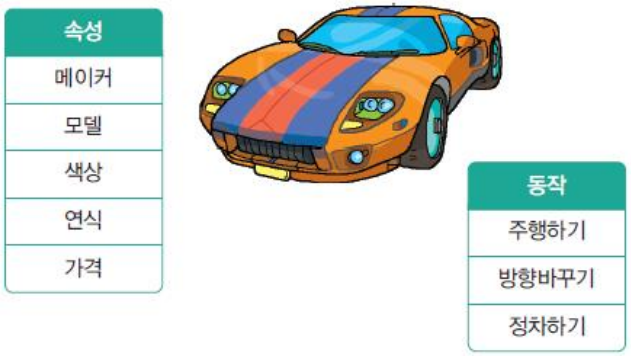
- 절차 지향 프로그래밍(procedural programming) :
  - 프로시저(procedure=함수)를 기반으로 하는 프로그래밍 방법
  - 단점 : 서로 관련된 데이터와 함수를 묶을 수가 없다. 절차 지향 방법에서는 데이터가 프로그램의 중요한 부분임에도 불구하고 프로그래머들은 함수 작성에만 신경을 쓰게 된다.
- 객체 지향 프로그래밍(object-oriented programming) : 데이터와 함수를 하나의 덩어리(캡슐화)로 묶어서 생각하는 방법





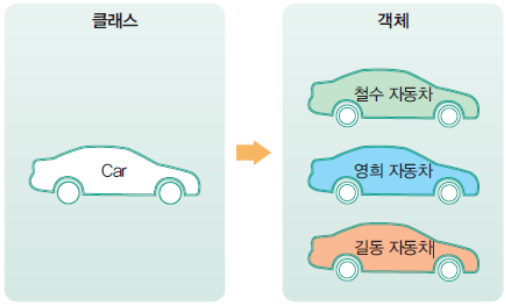
객체란. p419

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



클래스. p419

- 클래스(class)
  - 객체에 대한 설계도
  - 특정한 종류의 객체들을 찍어내는 형틀(template) 또는 청사진(blueprint)

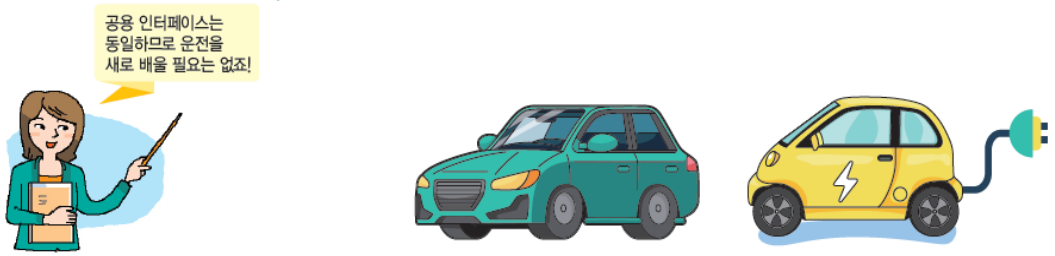


- 프로그램에서는 같은 종류의 객체가 많이 필요하기 때문에(예. 슈팅 게임에서 미사일들) 객체를 하나씩 정의하여 생성하는 것 보다, 클래스를 만들어 두고 필요할 때 마다 객체를 생성한다.
- 인스턴스(instance) : 클래스로부터 만들어지는 객체

파이썬에서는 모든 것이 객체이다. p420

- 공용 인터페이스(public interface) : 클래스에 의해 제공되는 메소드
- 개발자로서 클래스의 객체를 가지고 작업할 때는 내부적으로 속성을 저장하고 어떻게 메소드들이 구현되는지 알 필요가 없다.
- 캡슐화(encapsulation) : 공용 인터페이스만 제공하고 구현 세부 사항을 감추는 것

- 우리가 자동차를 구입하면 자동차가 어떻게 동작되는지 알 필요가 있을까? 우리에게 중요한 것은 자동차를 사용하는 방법이다.



클래스 작성하기. p422

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
```

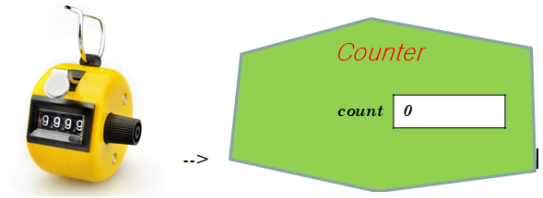
생성자를 정의한다.

메소드를 정의한다.

- 인스턴스 변수 : 클래스 안에 정의된 변수. ‘self.’을 붙이고 저장
- 메소드(method) : 클래스 안에 정의된 함수
- 멤버(member) : 인스턴스 변수와 메소드

클래스 작성하기. p422

- Counter() 클래스 만들기




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

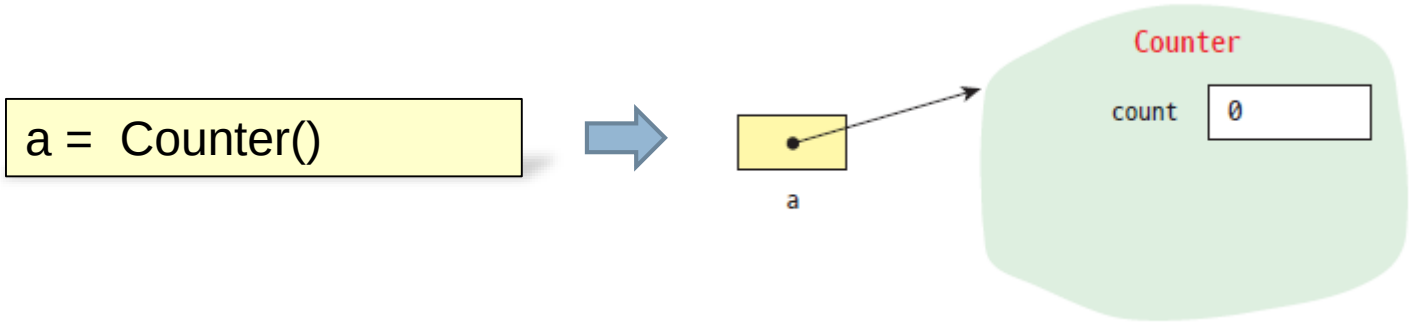
생성자 정의

인스턴스 변수 생성

- 클래스 안에서 생성자와 메소드를 정의
- 생성자(constructor) 메소드 :
  - __init()__
  - 객체가 생성되면 반드시 호출되어 객체를 초기화하는 함수. 인스턴스 변수를 생성

객체 생성. p423

- 클래스 이름에 ()를 붙여서 함수처럼 호출



객체의 멤버 접근. p424

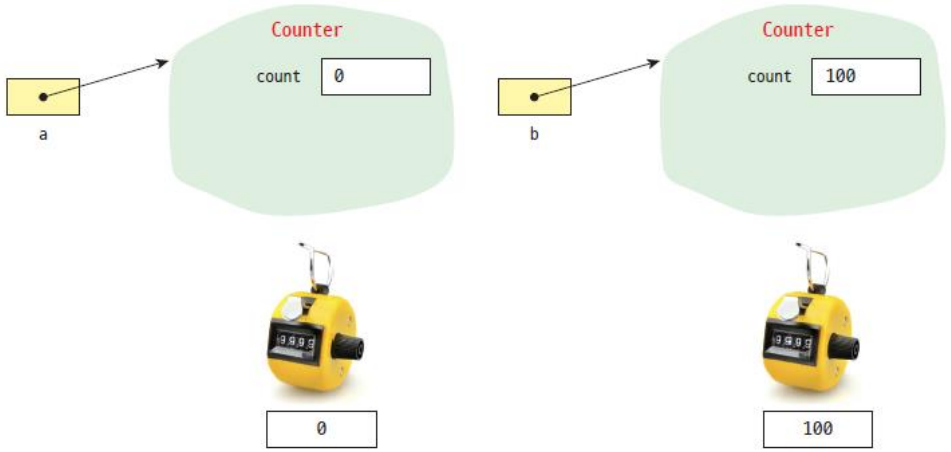
- 객체 이름에 점(.)을 붙이고 메소드 이름 또는 변수를 적어서 접근

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

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

```
카운터의 값= 1
```

하나의 클래스로 객체는 많이 만들 수 있다. p424

```
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.
```



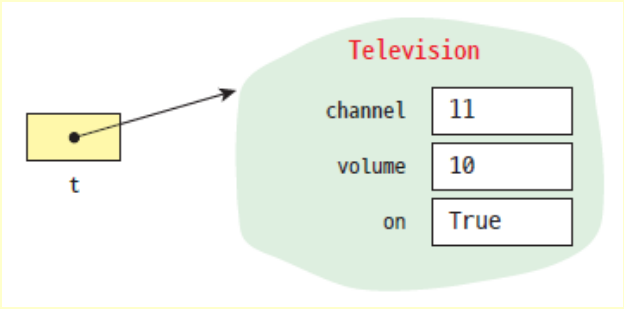
참고사항: 변수의 종류. P426

- 지역변수 : 함수 안에서 선언되는 변수
- 전역변수 : 함수 외부에서 선언된 변수
- 인스턴스 변수 : 클래스 안에서 선언된 변수. 앞에 self. 가 붙는다.

Lab TV 클래스 정의. p427

- 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)

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

        def getChannel(self):
                return self.channel
t = Television(9, 10, True)
t.show()
t.setChannel(11)
t.show()
```

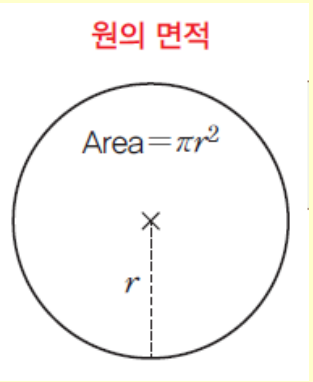



```
9 10 True
11 10 True
```

Lab 원 클래스 작성. p428

- 클래스 이름은 Circle로 하자. 원을 초기화하는 생성자는 만들어야 한다. 원은 반지름을 속성으로 가진다. 메소드로는 원의 넓이와 둘레를 반환하는 getArea()와 getPrimeter()를 정의한다.

```
import math

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 면적", c.getPerimeter())
```




```
원의 면적 314.1592653589793
원의 둘레 62.83185307179586
```

정보 은닉. p429

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

obj=Student("Hong", 20)
obj.age = 21
print(obj.age)
```

21

- 객체가 인스턴스 변수의 값을 클래스 바깥에서 직접 변경할 수 있는 것은 다음과 같은 이유에서 좋은 방법이 아니다.
  - 인스턴스의 값이 올바르지 않게 변경될 수 있다.
  - 클래스를 유지 보수 하는 것이 어려워진다.

정보 은닉. p429

- 정보 은닉(information hiding) : 클래스 안의 데이터를 외부에서 마음대로 변경하지 못하게 하는 것
- 변수 이름 앞에 __을 붙여 인스턴스 변수를 private으로 만들면 된다.

```
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. p431

- private로 정보 은닉된 인스턴스 변수를 외부에서 필요한 경우 접근할 수 있는 방법
- 접근자(getters) : 인스턴스 변수값을 반환하는 메소드
- 설정자(setters) : 인스턴스 변수값을 설정하는 메소드

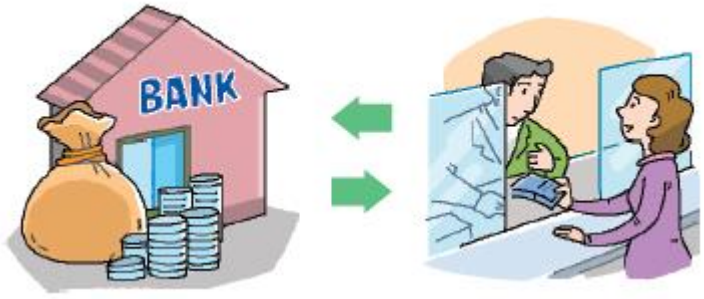
접근자와 설정자. p431

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

        def getAge(self):                # 접근자
                return self.__age

        def getName(self):
                return self.__name

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

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

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

Lab 은행 계좌. p433

- 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
```

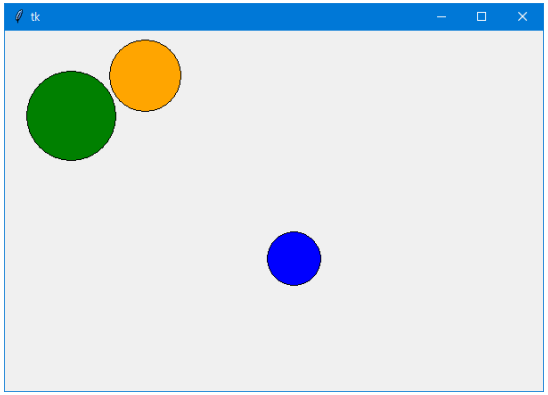


Lab: 은행 계좌. p433

```
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)
```

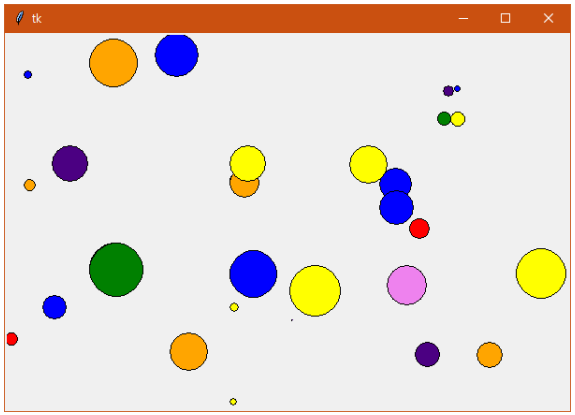
Lab 공 애니메이션 1. p434

- 실제로 객체 지향 기법이 많이 사용되는 곳이 GUI 응용 프로그램이다. 클래스를 정의해야만 GIU에서 나타나는 정보를 객체로 묶을 수 있기 때문이다. 크기와 색상이 다른 3개의 공이 캔버스에서 반사되는 응용 프로그램을 작성해보자.



Lab 공 애니메이션 2. p440

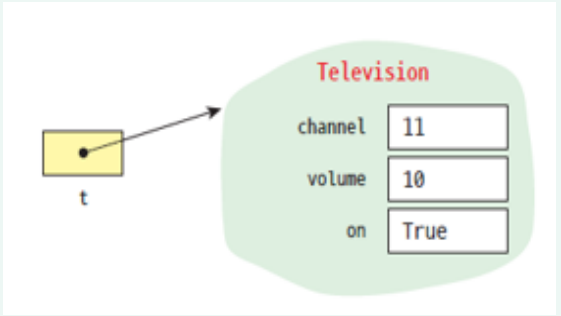
- 앞 절의 실습에서 공을 3개 생성하여서 화면에서 움직이게 하였다. 그런데 공을 30개 정도 만들어서 움직이려면 어떻게 해야 할까? 이럴 때는 개별 변수를 사용하여 각 객체를 참조하는 것은 거의 불가능하다. 이런 경우에는 리스트를 생성하고 리스트에 객체를 저장하여야 한다.



객체 참조. p436

- 변수는 단지 객체의 참조값(주소)을 저장한다. 객체 자체는 히프 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)
```

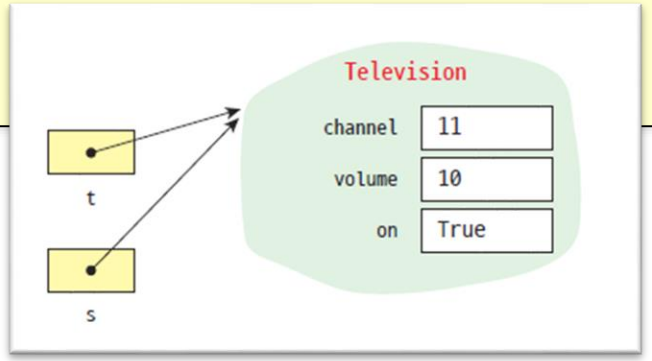


- 즉, 변수 t는 객체를 저장하고 있는 것이 아니라 객체의 참조값만 저장하고 있다.

참조 공유. p436

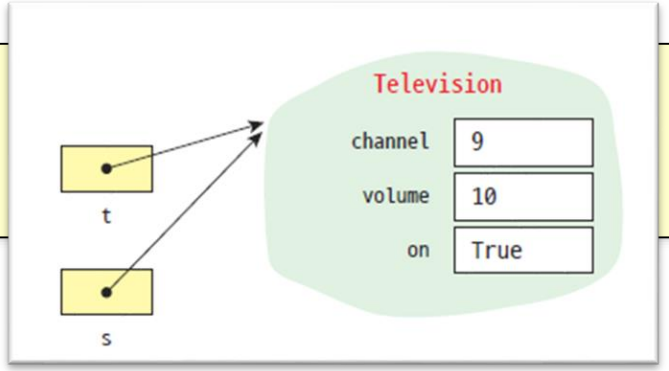
- 객체의 참조값을 저장하고 있는 변수를 다른 변수로 복사하면 어떻게 될까? 객체의 참조값만 복사되어 변수 s에 저장된다 → 얕은 복사

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



- s를 통하여 객체를 수정하면 어떻게 될까? t가 가리키는 객체의 값도 변경된다.

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



참조 공유. p437

- 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 참조값 p438

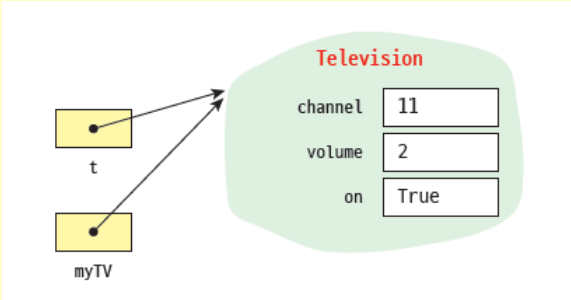
- None : 변수가 아무것도 참조하고 있지 않다는 것을 나타내는 특별한 값

```
myTV = None

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

객체를 함수로 전달할 때. p438

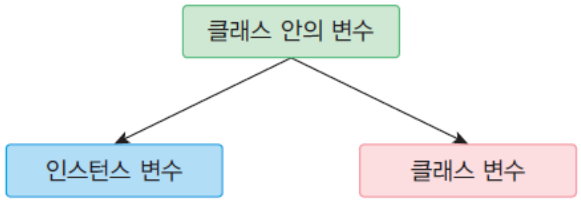
- 함수에 객체를 전달하면 객체의 참조값이 전달된다.

```
# 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()
```

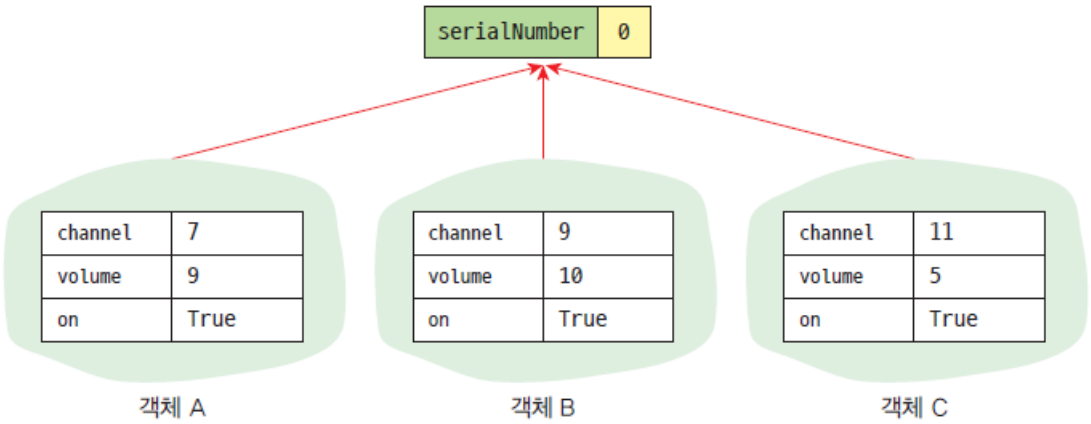



```
11 2 True
```

클래스 변수. p442



- 인스턴스 변수(instance variable) : 객체가 생성될 때 인스턴스 마다 별도로 생성되는 변수
- 클래스 변수(class variable) : 하나의 클래스에 하나만 존재. 모든 객체가 공유하는 변수



인스턴스 변수 vs 클래스 변수. p443

```
# 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. p443

- 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
```

특수 메소드. p444

- 객체에 대하여 +, -, *, / 와 같은 연산을 적용하면 자동으로 호출되는 메소드

```
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. p444

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

__str__() 메소드. p445

- __str__() 메소드는 객체를 print()로 출력할 때 자동으로 호출된다.

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

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

```
카운트값: 100
```

상속. p446

- 상속(inheritance) : 기존의 클래스로부터 변수와 메소드를 상속받아서 새로운 클래스를 파생하는 메커니즘
- 필요하다면 부모 클래스로 부터 상속받은 메소드를 교체하거나, 새로운 변수나 메소드를 추가할 수 있다.

상속. p446

```
class Person():
    def __init__(self, name):
        self.name = name
    def getName(self):
        return self.name
    def isStudent(self):
        return False

class Student(Person):
   def __init__(self, name, gpa):
       super().__init__(name)
       self.gpa = gpa

   def isStudent(self):    # 메소드 오버라이딩
        return True

obj1 = Person("Kim")
print(obj1.getName(), obj1.isStudent())

obj2 = Student("Park", 4.3)
print(obj2.getName(), obj2.isStudent())
```

Kim False
Park True

추가: 다중상속

- p447-01-apndx.py

추가: 메소드 오버라이딩

- 추가 내용. p447-02-apndx.py

Lab 특수 메소드. p448

- 2차원 공간에서 벡터(vector)는 (a, b)와 같이 2개의 실수로 표현될 수 있다. 벡터 간에는 덧셈이나 뺄셈이 정의된다. 특수 메소드를 이용하여서 + 연산과 – 연산, str() 메소드를 구현해보자.

```
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)

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

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

Mini Project 주사위 클래스 만들기. p449

- 주사위 클래스를 제공하는 파이썬 코드를 작성하고 테스트해보자.

Mini Project 주사위 클래스 만들기. p449

- 주사위 클래스를 제공하는 파이썬 코드를 작성하고 테스트해보자.

추가: 모듈

- 모듈(module) : 함수나 변수 또는 클래스를 모아 놓은 파일
- 예제. module 폴더

연습문제. p451

Programming. p453