산업 인공지능 - 실습 5

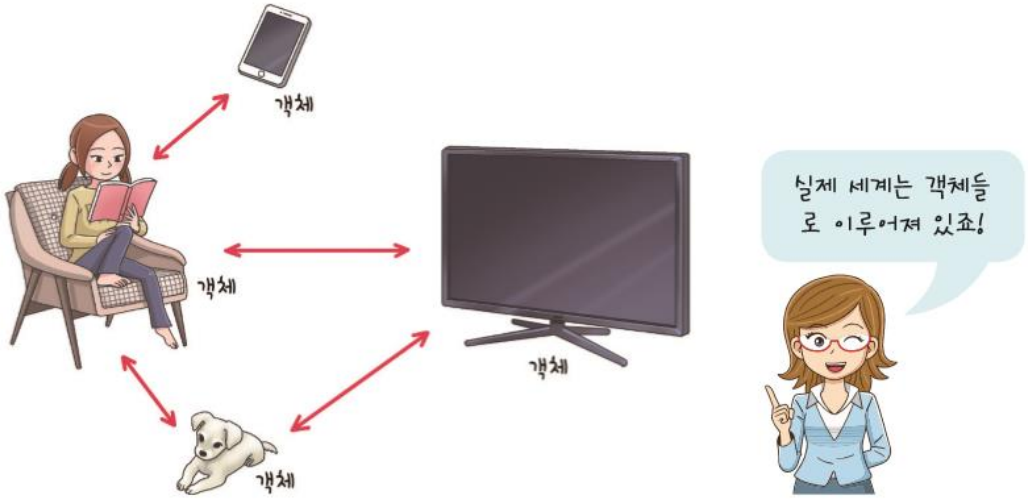
Python 프로그래밍

2021 Spring

1. 객체지향 프로그래밍

❖ **객체 지향 프로그래밍(OOP: object-oriented programming)**

- 프로그램을 여러 개의 **독립된 단위**(**객체, object**)**들의 모임**으로 구성
- **객체**는 처리할 대상에 대한 **데이터**와 데이터를 처리하는 **메소드**(method, function)들로 구성
- **객체를** 생성하고 **객체들의 메소드를 호출**하여, 프로그램을 기술하는 프로그래밍 패러다임(paradigm)
- 객체는 **재사용(reuse)**이 쉬워서, 일단 개발되면 다른 프로그램 개발에도 활용 가능
- 실세계의 일을 보다 쉽게 프로그래밍하는 것 가능



객체지향 프로그래밍

- **객체(object)**
  - 상태(state)와 동작(behavior) 보유
  - **생태**
    - 객체의 속성
    - 데이터 저장 : 인스턴스 변수(instance variable)
  - **동작**
    - 객체가 할 수 있는 동작(기능)
    - 메소드(method)

객체지향 프로그래밍

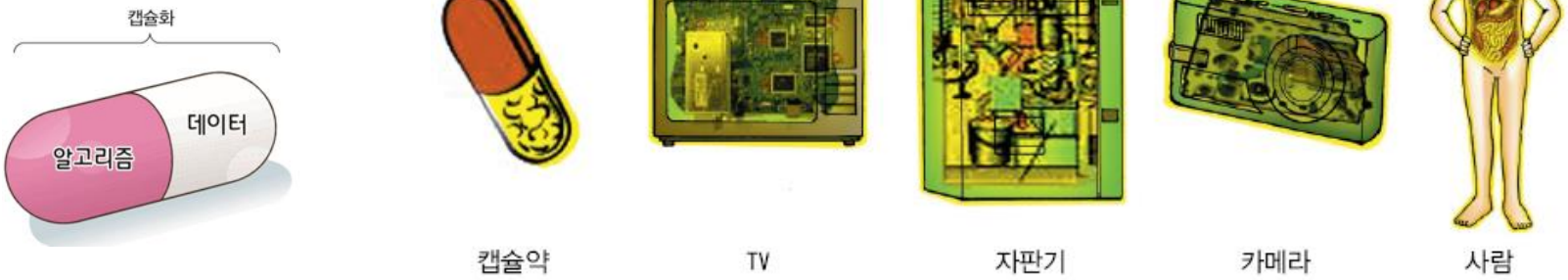
❖ **클래스(class)**
- 객체를 생성할 때 사용하는 설계도
- 특정 종류의 객체를 찍어내는 형틀(template) 또는 청사진(blueprint)

❖ **객체(object)**
- 클래스를 통해 생성된 대상

객체지향 프로그래밍

❖ 캡슐화 (encapsulation)

- **메소드(함수)와 데이터**를 **클래스 내**에 선언하고 구현
- 외부에서는 **공개된 메소드의 인터페이스**만 **접근** 가능
  - 외부에서는 **비공개 데이터**에 직접 접근하거나 **메소드**의 구현 세부를 알 수 없음
- 객체 내 데이터에 대한 보안, 보호, 외부 **접근 제한**



실세계의 캡슐화

객체지향 프로그래밍

❖ **파이썬의 객체**

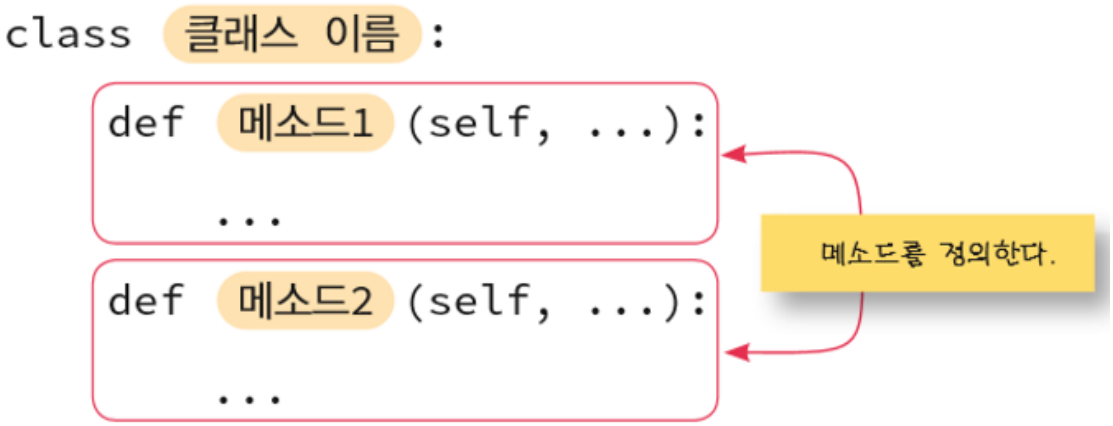
- **모든 것**이 **객체**
  - 정수, 문자열, 리스트 등
- 객체는 메소드 제공
  - 메소드를 통해 객체 사용

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

>>> (1).__add__(2)
3
```

2. 클래스

❖ 클래스(class)

- 객체의 형태를 정의하는 틀(template)

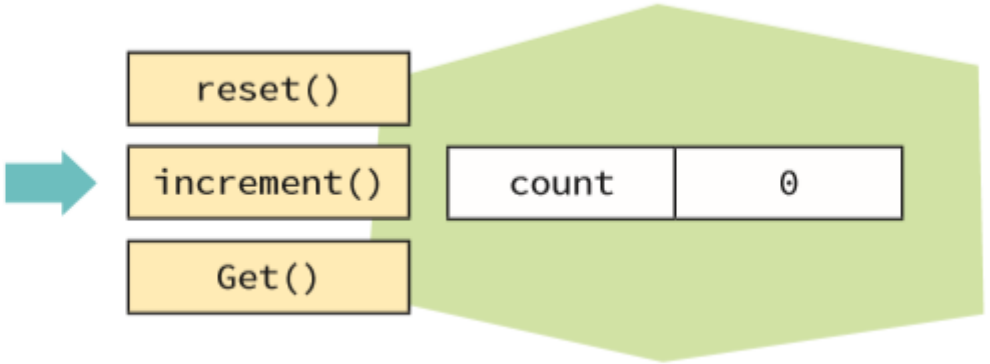


- 클래스의 **멤버**(member)
  - **인스턴스 변수**(instance variable, **필드**; field)
    - **객체 안에 정의한 변수** : **self.** 를 붙인 변수에 값 대입
    - 객체의 상태 표현
  - **메소드**(method)
    - 객체의 동작

클래스

❖ 예: **Counter 클래스**

- 기계식 계수기




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

인스턴스 변수 생성

별도 선언없이
초기화하면 생성

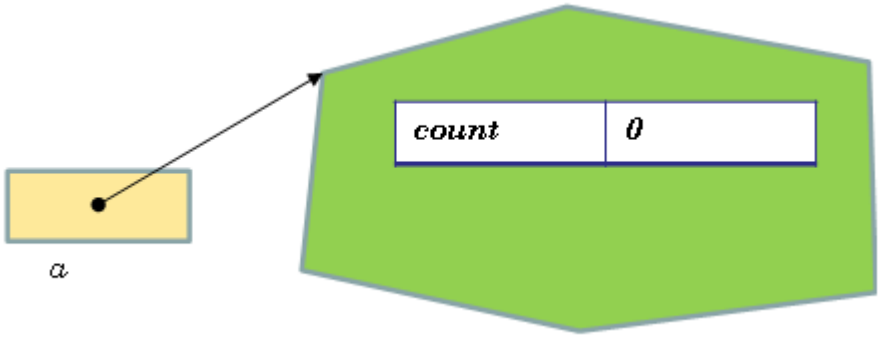
self : 메소드를 호출한 객체를
가리키는 참조변수

클래스

❖ 객체 생성

```
a = Counter()

a.reset()
a.increment()
print("카운터 a의 값은", a.get())
```

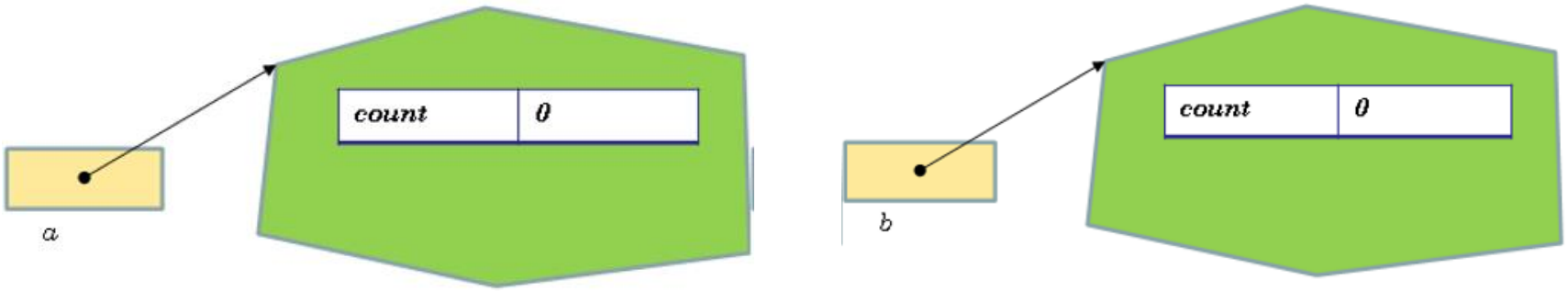
```
카운터 a의 값은 1
```



클래스

❖ 여러 개의 객체 생성

```
a = Counter()
b = Counter()

a.reset()
b.reset()
```



3. 생성자

❖ 생성자(constructor)

- 객체가 **생성**될 때 **자동**으로 **호출**되는 특수한 메소드
- 객체를 기본값으로 초기화하는 역할
- __init__(self) 메소드로 정의
  - self : 현재 초기화되는 객체를 가리키는 참조변수

```
class 클래스 이름:

    def __init__(self, ...):

        ...
```

__init__() 메소드가 생성자이다.
여기서 객체의 초기화를 담당한다.

- 클래스별로 하나의 생성자 정의 가능
- 생성자에 여러 매개변수 허용

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

객체 생성시 값을 전달하지 않으면 0으로 간주

```
a = Counter(100)
b = Counter()
```

생성자

❖ 생성자 정의의 예

```
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
```

4. 메소드

❖ **메소드(method)**

- 클래스 안에 정의된 함수
- **첫 번째 매개변수**는 항상 **self**

```
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
```

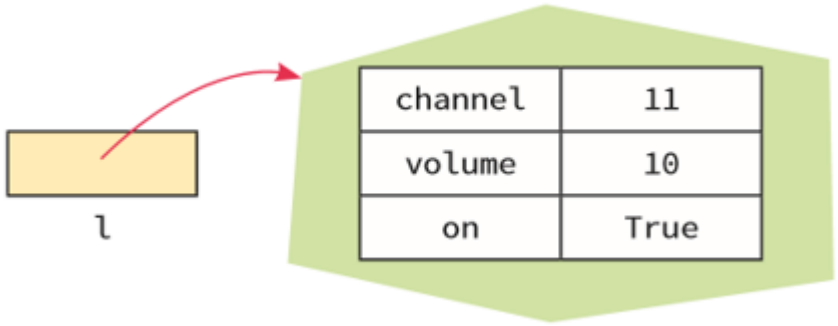
메소드

❖ 메소드 호출

- 생성된 객체에 대해 **멤버 연산자(.)**를 사용하여 호출

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

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

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



메소드

❖ self

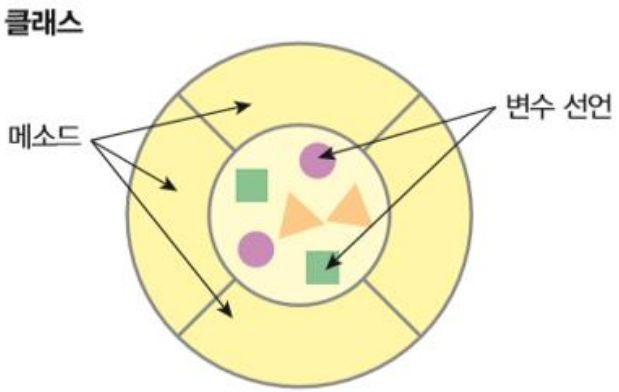
- 객체 자신을 참조하는 변수
- self.을 사용하여 객체의 멤버 접근
  - 인스턴스 변수 접근 : **self.변수명**
  - 메소드 접근 : **self.메소드명**

- **인스턴스 변수의 범위**
  - **__init__( ) 메소드 내**에서 정의 : **클래스 전체**
  - **메소드 안**에서 생성된 변수 : **지역 변수**, 메소드 내에서만 사용

5. 정보 은닉

❖ **정보 은닉(information hiding)**

- 클래스 구현의 세부 사항을 클래스 안에 감추는 것
- 공개된 변수와 메소드를 사용하여 클래스 사용



- **private 멤버**
  - 클래스 내부에서만 사용할 수 있는 변수 또는 메소드
  - 이름 __ 로 시작하는 변수 또는 메소드

정보 은닉

❖ **private 멤버 사용 예**

```
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'
```

6. 접근자와 설정자

❖ **접근자(getter)**
- 인스턴스 변수값을 **반환**하는 메소드
- private 인스턴스 변수 접근에 사용

❖ **설정자(setter)**
- 인스턴스 변수값을 **설정**하는 메소드
- private 인스턴스 변수 값 설정에 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("Hong", 20)
obj.getName()
```

접근자와 설정자

❖ **접근자와 설정자 사용 이유**

- 나중에 클래스 업그레이드시 편리
- 매개 변수를 통해서 잘못된 값이 넘어올 때 사전에 처리 가능
- 필요할 때마다 인스턴스 변수 값을 계산하여 반환 가능
- 접근자만 제공하면 읽기만 할 수 있는 인스턴스 변수 생성 가능

예: 원 클래스

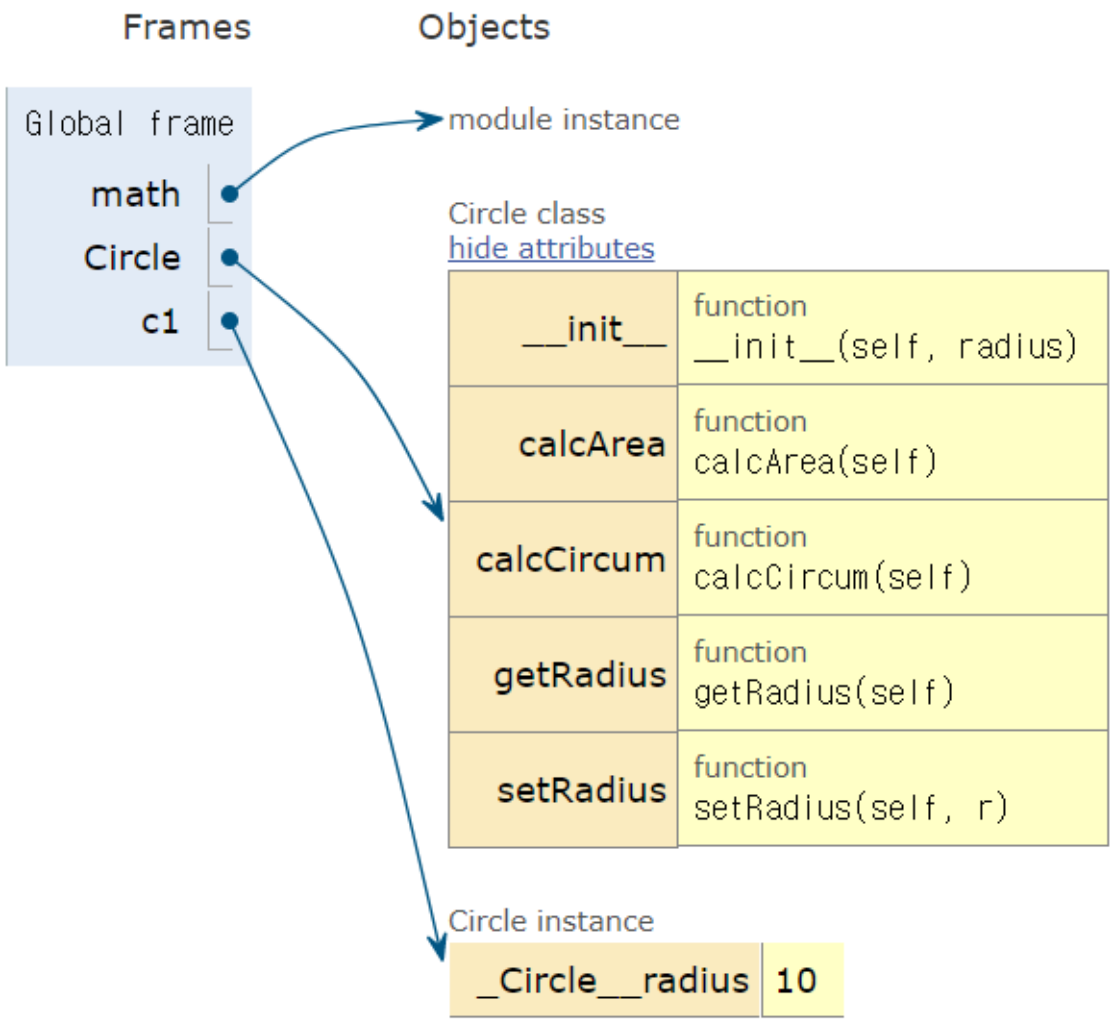
❖ **원(Circle) 클래스**
- 반지름 데이터 저장
- 넓이, 둘레 계산 메소드 제공
- 데이터에 대한 접근자와 설정자 제공

```
원의 반지름= 10
원의 넓이= 314.1592653589793
원의 둘레= 62.83185307179586
```

원 클래스

```
import math
class Circle:
    def __init__(self, radius=1.0):
        self.__radius = radius

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

    def getRadius(self):
        return self.__radius

    def calcArea(self):
        area = math.pi*self.__radius*self.__radius
        return area
    def calcCircum(self):
        circumference = 2.0*math.pi*self.__radius
        return circumference

c1 = Circle(10)
print("원의 반지름=", c1.getRadius())
print("원의 넓이=", c1.calcArea())
print("원의 둘레=", c1.calcCircum())
```

원 클래스


Frames
Objects
Global frame
math
Circle
c1
module instance
Circle class
hide attributes
__init__
function
__init__(self, radius)
calcArea
function
calcArea(self)
calcCircum
function
calcCircum(self)
getRadius
function
getRadius(self)
setRadius
function
setRadius(self, r)
Circle instance
_Circle__radius
10

7. 객체의 함수 전달

- **객체의 참조변수 함수 전달**
  - 객체의 내용 변경 가능

- **수와 문자열 전달**
  - 값 자체가 전달되므로 원본 변경 불가

```
class Rectangle:
    def __init__(self, side=0):
        self.side = side

    def getArea(self):
        return self.side*self.side

def printAreas(r, n):
    while n >= 1:
        print(r.side, "\t", r.getArea())
        r.side = r.side + 1
        n = n - 1
```

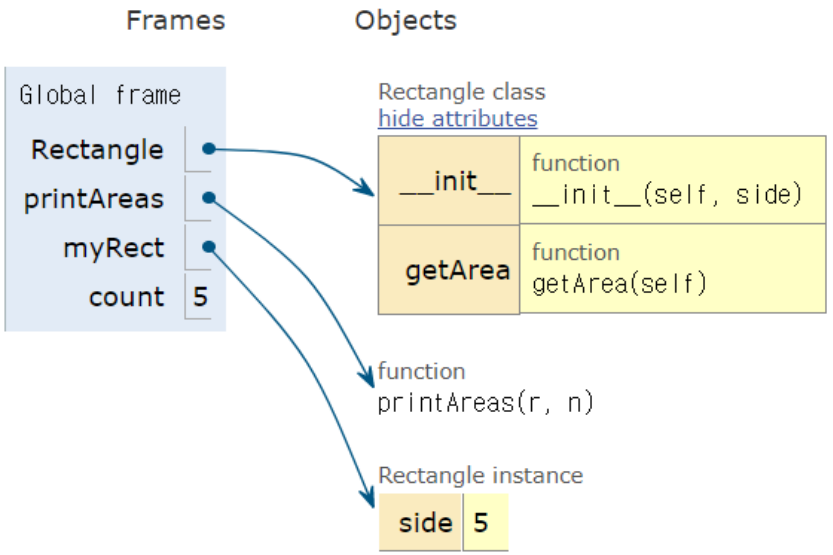
```
myRect = Rectangle();
count = 5
printAreas(myRect, count)
print("사각형의 변=", myRect.side)
print("반복횟수=", count)
```

```
0       0
1       1
2       4
3       9
4       16
사각형의 변= 5
반복횟수= 5
```

객체의 함수 전달

```
class Rectangle:
    def __init__(self, side=0):
        self.side = side

    def getArea(self):
        return self.side*self.side

def printAreas(r, n):
    while n >= 1:
        print(r.side, "\t", r.getArea())
        r.side = r.side + 1
        n = n - 1


myRect = Rectangle();
count = 5
printAreas(myRect, count)
print("사각형의 변=", myRect.side)
print("반복횟수=", count)
```

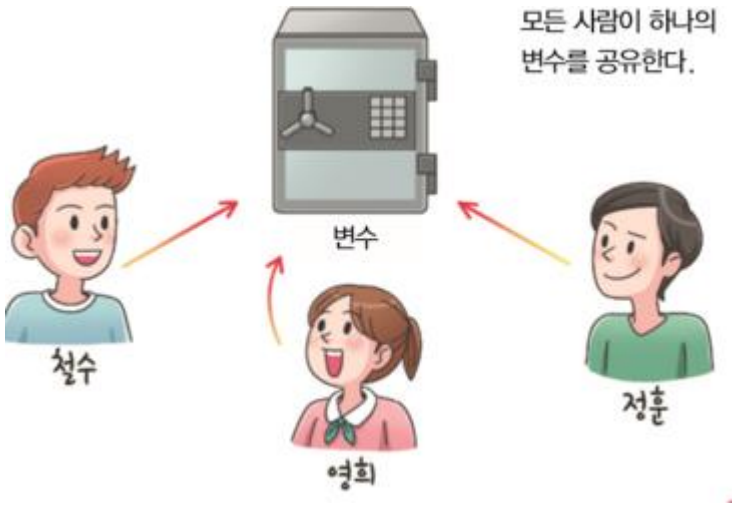
```
0       0
1       1
2       4
3       9
4       16
사각형의 변= 5
반복횟수= 5
```



8. 클래스 변수

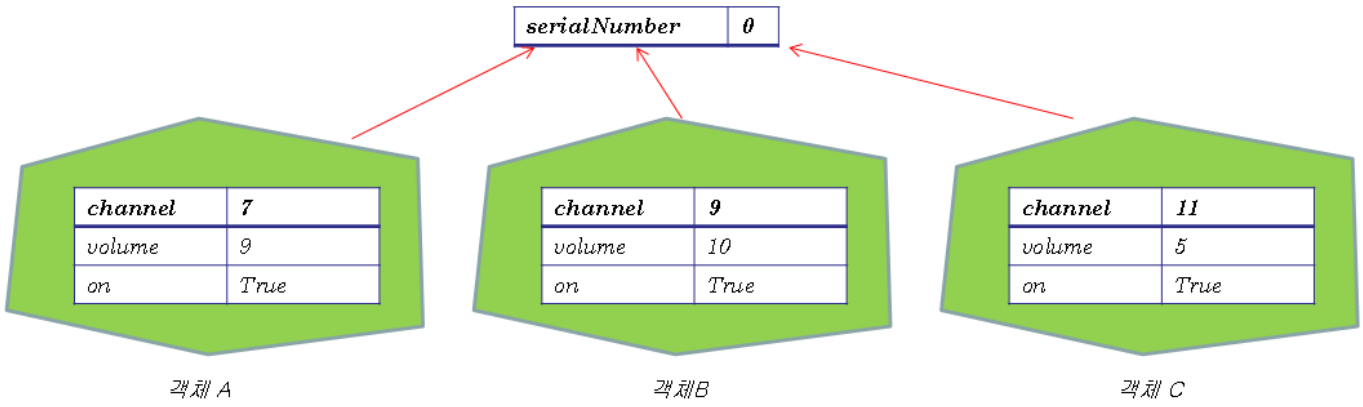
❖ **인스턴스 변수(instance variable)**
- 객체 마다 별도로 생성

❖ **클래스 변수(class variable) 또는 정적 변수**
- 동일 클래스의 모두 객체가 공동으로 사용
- 클래스 마다 한 개만 생성
- 클래스 안에 있지만 메소드 외부에 생성 : **self 미사용**



클래스 변수

```
class Television:
    serialNumber = 0          #  클래스 변수
    def __init__(self):
        Television.serialNumber += 1

    self.number = Television.serialNumber
        ...
```



9. 특수 메소드

❖ **특수 메소드(special method)**

- 연산자(+, -, *, /)에 관련된 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. ")
```

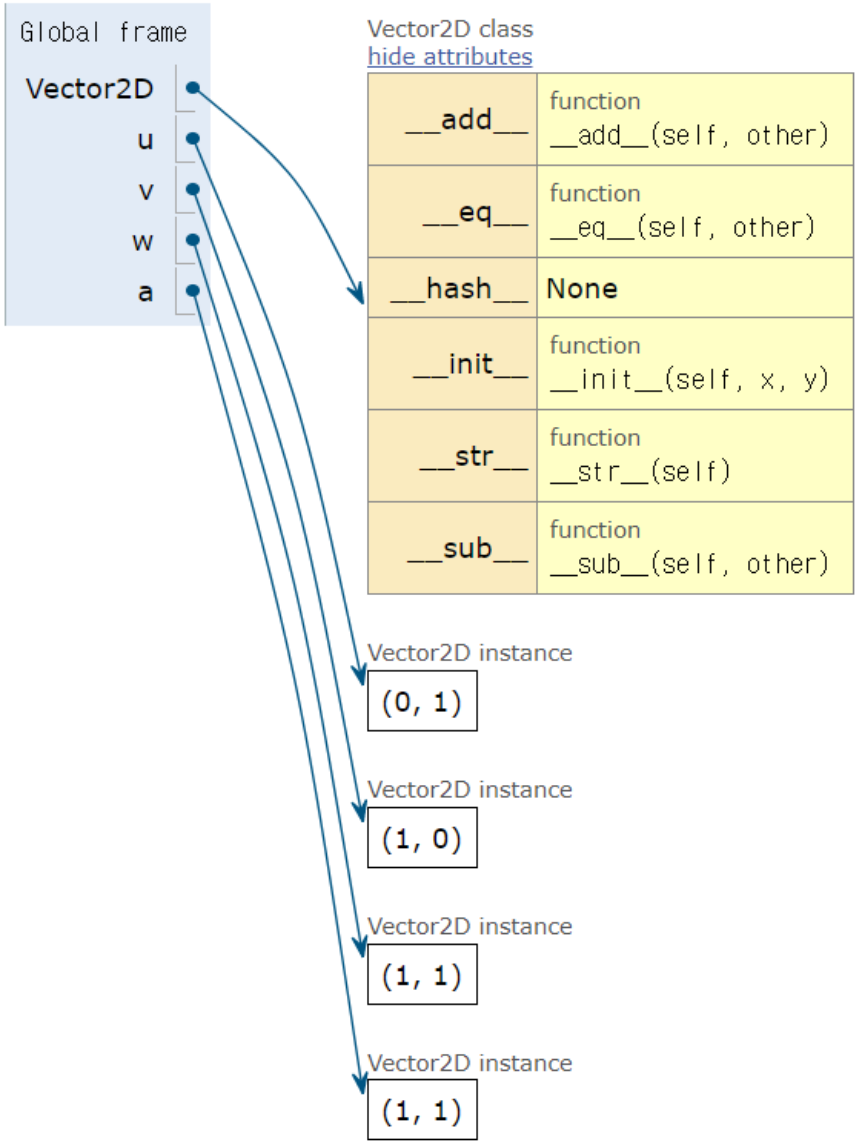
특수 메소드

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

```
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)
```

특수 메소드


Global frame
Vector2D
u
v
w
a
Vector2D class
hide attributes
__add__
function
__add__(self, other)
__eq__
function
__eq__(self, other)
__hash__
None
__init__
function
__init__(self, x, y)
__str__
function
__str__(self)
__sub__
function
__sub__(self, other)
Vector2D instance
(0, 1)
Vector2D instance
(1, 0)
Vector2D instance
(1, 1)
Vector2D instance
(1, 1)

10. 변수의 종류

❖ **변수의 종류**

- 지역변수
  - 함수 안에서 선언되는 변수

- 전역변수
  - 함수 외부에서 선언되는 변수

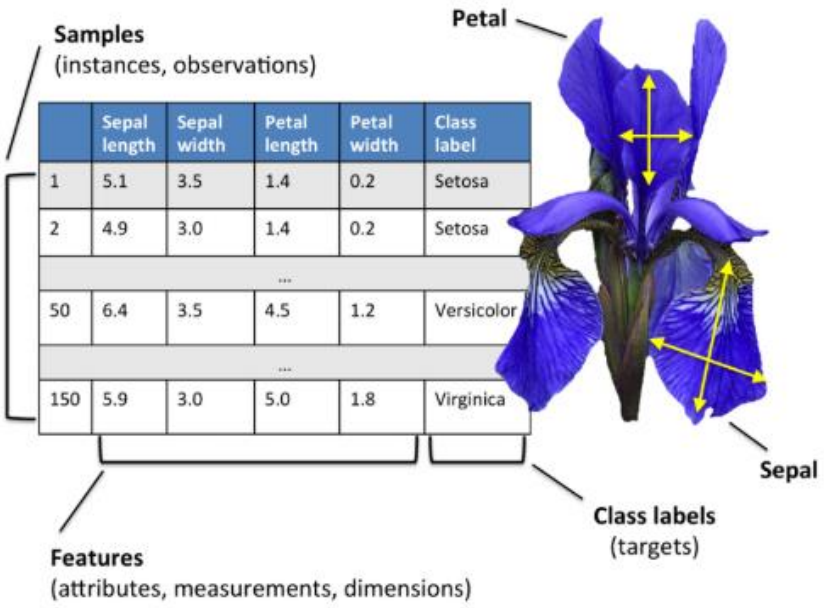
- 인스턴스 변수
  - 클래스 안에서 선언되는 변수
  - 앞에 self.가 붙음

분류, 회귀

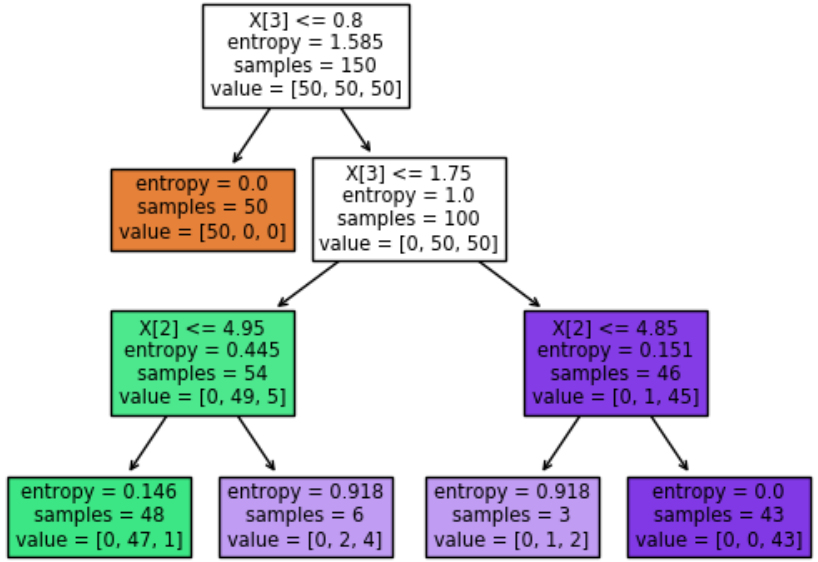
분류

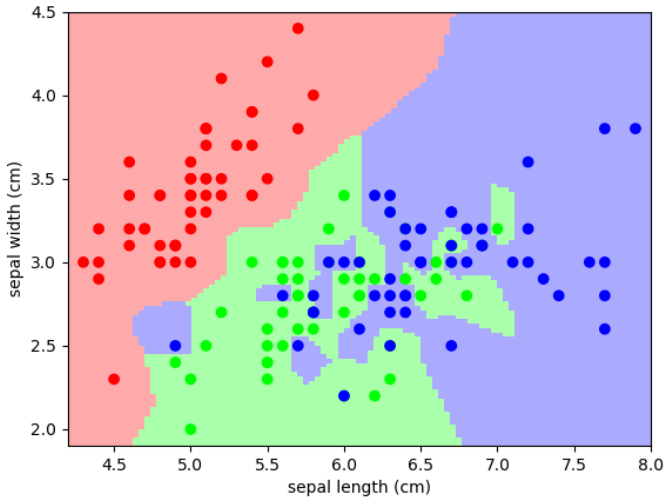
❖ 분류 학습 데이터와 학습 결과의 예

- 수치형 속성



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



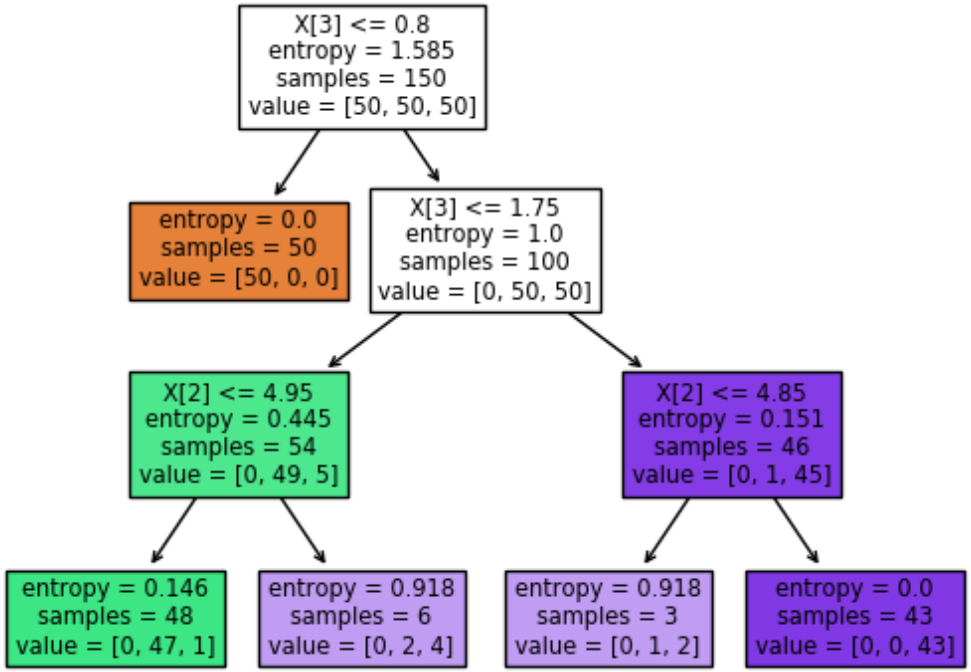


https://medium.com/@jebaseelanravi96/machine-learning-iris-classification-33aa18a4a983

https://pythonmachinelearning.pro/supervised-learning-using-decision-trees-to-classify-data/

분류

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

iris = load_iris()
decision_tree = DecisionTreeClassifier(criterion="entropy", random_state=0, max_depth=3)
decision_tree = decision_tree.fit(iris.data, iris.target)

plt.figure()
plot_tree(decision_tree, filled=True)
plt.show()
```

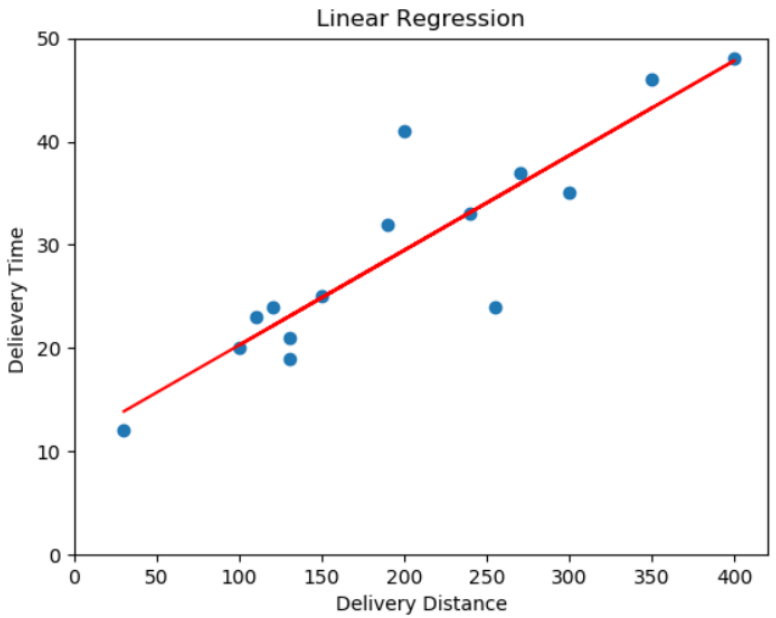


회귀

❖ [실습] 선형회귀

- 입력: 배달거리(delivery distance), 출력: 배달시간(delivery time)
- 파라미터에 대한 1차 방정식을 사용한 회귀

```
import numpy as np
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot as plt

data = np.array([[30, 12], [150, 25], [300, 35], [400, 48], [130, 21],
                 [240, 33],[350, 46], [200, 41], [100, 20], [110, 23],
                 [190, 32], [120, 24], [130, 19], [270, 37], [255, 24]])

plt.scatter(data[:, 0], data[:, 1])  # 데이터 위치의 산포도 출력
plt.title("Linear Regression")
plt.xlabel("Delivery Distance")
plt.ylabel("Delievery Time ")
plt.axis([0, 420, 0, 50])

x = data[:, 0].reshape(-1, 1)   # 입력
y = data[:, 1].reshape(-1, 1)   # 출력

model = LinearRegression( )
model.fit(x, y)                 # 모델 학습

y_pred = model.predict(x)   # 예측값 계산
plt.plot(x, y_pred, color='r')
plt.show( )
```



[실습] 로지스틱 회귀

https://drive.google.com/file/d/1Upqoz2gIAYq6LByD7YjHU-9_9K4EOhOh/view

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('User_Data.csv')

x = dataset.iloc[:, [2, 3]].values   # 입력
y = dataset.iloc[:, 4].values   # 출력

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)
print (xtrain[0:10, :])

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
y_pred = classifier.predict(xtest)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, y_pred)
print ("혼동행렬 : ₩n", cm)

from sklearn.metrics import accuracy_score
print ("정확도 : ", accuracy_score(ytest, y_pred))
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]]
혼동행렬 :
 [[65  3]
 [ 8 24]]
정확도 :  0.89
```

https://www.geeksforgeeks.org/ml-logistic-regression-using-python/

```
from matplotlib.colors import ListedColormap
X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(
             np.array([X1.ravel(), X2.ravel()]).T).reshape(
             X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```