**교육일지**

**교육 제목 : class, 판다스입문(시리즈)**

**교육 장소 : YGL C6 강의실**

**교육 일시 : 2021/09/28**

**오전 교육**

# class 와 object

classs는 객체를 쉽고 편리하게 만들기 위해 만들어진 구문

# 학생 리스트를 선언합니다.

students = [

{ "name": "윤인성", "korean": 87, "math": 98, "english": 88, "science": 95 },

{ "name": "연하진", "korean": 92, "math": 98, "english": 96, "science": 98 },

{ "name": "구지연", "korean": 76, "math": 96, "english": 94, "science": 90 }, ]

# 딕셔너리를 리턴하는 함수를 선언

def create\_student(name,korean,math,english,science):

return {

"name":name, "korean":korean, "english":english,"math":math,"science":science}

#학생 점수의 합을 구하는 함수

def student\_get\_sum(student):

return student["korean"]+student["math"] + \

student["english"]+student["science"]

#평균을 구하는 함수

def student\_get\_avg(student):

return student\_get\_sum(student)/4

#출력하는 함수

def student\_to\_string(student):

return "{}:\t{}\t{}".format(student["name"],student\_get\_sum(student),

student\_get\_avg(student))

#class로 생성하여 object를 관리

class Student: # class 클레스이름:

def \_\_init\_\_(self,name,korean,math,english,science):

self.name = name self.korean = korean self.math = math self.english = english

self.science = science

def get\_sum(self):

return self.korean+self.math+self.english+self.science \

def get\_avg(self):

return self.get\_sum()/4

def to\_string(self):

return "{}:\t{}\t{}".format(self.name,\

self.get\_sum(),\

self.get\_avg())

def \_\_eq\_\_(self, value): #같다

return self.get\_sum() == value.get\_sum()

def \_\_ne\_\_(self, value): #다르다

return self.get\_sum() != value.get\_sum()

def \_\_gt\_\_(self, value): #크다

return self.get\_sum() > value.get\_sum()

def \_\_ge\_\_(self, value): #크거나 같다

return self.get\_sum() >= value.get\_sum()

def \_\_lt\_\_(self, value): #작다

return self.get\_sum() < value.get\_sum()

def \_\_le\_\_(self, value): #작거나 같다

return self.get\_sum() <= value.get\_sum()

a\_class = Student("윤인성", 87, 98, 88, 95)

#Student 클래스, a\_class 인스턴스, 데이터

print(a\_class.name,a\_class.math, a\_class.english)

**오후 교육**

판다스 (시리즈)

시리즈는 데이터가 순차적으로 나열된 1차원 배열의 형태를 말한다

딕셔너리와 시리즈는 구조가 비슷해서 딕셔너리를 시리즈로 변환하는 방법을 많이 사용한다. 예)

import pandas as pd #판다스 불러오기

dict\_data = {'a':1, 'b':2, 'c':3} #'a'는 key 값, 판다스 series() 함수로 dictionary를 series로 변환, 변수 sr에 저장

sr = pd.Series(dict\_data) # dictionary를 series 형식으로 변환

print(type(dict\_data))

print(type(sr))

print(dict\_data)

print(ar)

print(dict\_data['a']," : ", ar['a']," : ",ar[0]) # ar에서 'a' 인덱스명

print(ar[['a','c']])

ar[1:2] # 인덱스철자로 접근하면 마지막 첨자 위치 데이터 불포함

ar['b':'c'] # 인덱스명으로 접근하면 마지막 데이터 포함

# 딕셔너리를 시리즈로 변경 : 딕셔너리에 키가 시리즈의 인덱스명이 됨

# 접근은 인덱스명 또는 인덱스 첨자로 접근 가능

데이터프레임

데이터프레임은 2차원 배열, 엑셀과 관계형 데이터베이스 사용

#행인덱스 열이름 설정 : pandas.DateFrame(2차원 배열, index= 행 인덱스 배열,

columns= 열이름 배열)

#행인덱스명 변경 : object.index= 새로운 행 인덱스 배열

#컬럼 인덱스명 변경 : object.columns = 새로운 컬럼명 변겅

# 행삭제 axis= 0, 열 삭제 = axis =1

#행 인덱스 변경 : DateFrame 객체.rename(index={기존 인덱스:새인덱스})

# 열 이름 변경 : DateFrame 객체.rename(columns={기존 이름:새 이름})