|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 210928 |
| 교육장소 | 자택(디스코드 사용 온라인 학습) |
| **교육내용** | |
| 오전 | 1. 클래스   : 객체를 조금 더 효율적으로 생성하기 위해 만들어진 구문  class 클래스 이름 :  클래스 내용  인스턴스 이름(변수 이름) = 클래스 이름() -> 생성자 함수  객체 지향 프로그래밍 : 객체를 우선으로 생각해서 프로그래밍 하는 것  # class와 object  # 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 },  { "name": "나선주", "korean": 98, "math": 92, "english": 96, "science": 92 },  { "name": "윤아린", "korean": 95, "math": 98, "english": 98, "science": 98 },  { "name": "윤명월", "korean": 64, "math": 88, "english": 92, "science": 92 }  ]  Ex1)  # 학생을 한 명씩 반복  print("이름", "총점", "평균", sep='\t')  for student in students:  # 점수의 총합과 평균을 구함  sco\_sum = student["korean"]+student["math"]+student["english"] +\  student["science"]  sco\_avg = sco\_sum / 4  print(student["name"], sco\_sum, sco\_avg, sep='\t')  Ex2)  # 딕셔너리를 리턴하는 함수를 선언  def create\_student(name, korean, math, english, science):  return {  "name" : name,  "korean" : korean,  "english" : english,  "math" : math,  "science" : science  }  students = [  create\_student("윤인성", 87, 98, 88, 95),  create\_student("연하진", 92, 98, 96, 98),  create\_student("구지연", 76, 96, 94, 90),  create\_student("나선주", 98, 92, 96, 92),  create\_student("윤아린", 95, 98, 98, 98),  create\_student("윤명월", 64, 88, 92, 92)  ]  # 학생 점수의 합을 구하는 함수  def student\_get\_sum(student):  return student["korean"]+student["math"]+student["english"] +\  student["science"]  # 학생 점수의 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))   1. 생성자   클래스 이름과 같은 함수  class 클래스 이름 :  def \_ init \_(self, 추가적인 매개변수):  pass  ex)  # 클래스 내부에 함수 선언하기  # class로 생성하여 object를 관리  class student: # 함수가 아니니까 () 안넣어줌 // class 클래스 이름 :  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())    a\_class = student("윤인성", 87, 98, 88, 95)  #student-class//a\_class-인스턴스,데이터  print(a\_class.name, a\_class.math, a\_class.english)  print("이름", "총점", "평균", sep='\t')  for student in students:  print(student.to\_string())  # 인스턴트,클래스명  class human:  def \_\_init\_\_(self):  pass  class Student(human):  def \_\_init\_\_(self):  pass    student = Student()  #instance 확인  print("isinstance(student,Human) :", ininstance)   1. 메소드   : 클래스가 가지고 있는 함수.  class 클래스 이름:  def 메소드 이름(self, 추가적인 매개변수):  pass  ex)  class student: # 함수가 아니니까 () 안넣어줌 // class 클래스 이름 :  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 \_\_str\_\_(self):  return "{}:\t{}\t{}".format(self.name, self.get\_sum(), self.get\_avg())  def \_\_eq\_\_(self, value):  return self.get\_sum() == value.get\_sum()  def \_\_ne\_\_(self, value):  return self.get\_sum() != value.get\_sum()  def \_\_gt\_\_(self, value):  return self.get\_sum() > value.get\_sum()  def \_\_ge\_\_(self, value):  return self.get\_sum() >= value.get\_sum()  def \_\_lt\_\_(self, value):  return self.get\_sum() < value.get\_sum()  def \_\_le\_\_\_\_(self, value):  return self.get\_sum() <= value.get\_sum()  students = [  student("윤인성", 87, 98, 88, 95),  student("연하진", 92, 98, 96, 98),  student("구지연", 76, 96, 94, 90),  student("나선주", 98, 92, 96, 92),  student("윤아린", 95, 98, 98, 98),  student("윤명월", 64, 88, 92, 92)  ]  print("이름", "총점", "평균", sep='\t')  for student in students:  print(str(student)) # student의 \_\_str\_\_()함수가 실행 됨    print("students[0] == students[1] =", students[0] != students[1] )  class Student:  count = 0  def \_\_init\_\_(self, name, korean, math, english, science):  # 인스턴스 변수 초기화  self.name = name  self.korean = korean  self.math = math  self.english = english  self.science = science    # 클래스 변수 설정  Student.count += 1  print("{}번째 학생이 생성되었습니다.".format(Student.count))    # 학생 리스트를 선언합니다.  students = [  Student("윤인성", 87, 98, 88, 95),  Student("연하진", 92, 98, 96, 98),  Student("구지연", 76, 96, 94, 90),  Student("나선주", 98, 92, 96, 92),  Student("윤아린", 95, 98, 98, 98),  Student("윤명월", 64, 88, 92, 92)  ]  # 출력합니다.  print()  print("현재 생성된 총 학생 수는 {}명입니다.".format(Student.count))  # 클래스 내부와 외부에서 클래스 변수에 접근할 때는 모두  # student.count 형태(클래스이름.변수이름)를 사용함.   1. 클래스 함수   : 클래스가 가진 함수, 클래스가 가진 기능 명시적으로 나타냄  데코레이터 : @classmethod  클래스 함수 만들기  class 클래스 이름:  @classmethod  def 클래스 함수(cls, 매개변수):  pass  클래스 함수 호출하기  클래스이름.함수 이름(매개변수)  # 클래스를 선언합니다.  class Student:  # 클래스 변수  count = 0  students = []  # 클래스 함수  @classmethod  def print(cls):  print("------ 학생 목록 ------")  print("이름\t총점\t평균")  for student in cls.students:  print(str(student))  print("------- ------- -------")  # 인스턴스 함수  def \_\_init\_\_(self, name, korean, math, english, science):  self.name = name  self.korean = korean  self.math = math  self.english = english  self.science = science  Student.count += 1  Student.students.append(self)  def get\_sum(self):  return self.korean + self.math +\  self.english + self.science  def get\_avg(self):  return self.get\_sum()/4    def get\_average(self):  return self.get\_sum() / 4  def \_\_str\_\_(self):  return "{}\t{}\t{}".format(\  self.name,\  self.get\_sum(),\  self.get\_average())    # 학생 리스트를 선언합니다.  Student("윤인성", 87, 98, 88, 95),  Student("연하진", 92, 98, 96, 98),  Student("구지연", 76, 96, 94, 90),  Student("나선주", 98, 92, 96, 92),  Student("윤아린", 95, 98, 98, 98),  Student("윤명월", 64, 88, 92, 92)  #출력  Student.print()   1. 가비지 컬렉터   더 사용할 가능성이 없는 데이터를 메모리에서 제거하는 역할   1. 프라이빗 변수   변수를 마음대로 사용하는 것 방지  \_\_<변수 이름> 형태로 인스턴스 변수 이름 선언  Ex)  # 모듈을 가져옵니다.  import math  # 클래스를 선언합니다.  class Circle:  def \_\_init\_\_(self, radius):  self.\_\_radius = radius  def get\_circumference(self):  return 2 \* math.pi \* self.\_\_radius  def get\_area(self):  return math.pi \* (self.\_\_radius \*\* 2)  # 여기 내부에서만 \_\_radius 접근가능  def get\_radius(self): # getter  return self.\_\_radius  def set\_radius(self, value): # setter  self.\_\_radius = value  # 원의 둘레와 넓이를 구합니다.  circle = Circle(10)  print("# 원의 둘레와 넓이를 구합니다.")  print("원의 둘레:", circle.get\_circumference())  print("원의 넓이:", circle.get\_area())  print()   1. Setter와 Getter   # \_\_radius에 접근합니다.  print("# \_\_radius에 접근합니다.")  print(circle.get\_radius())# 간접적으로 접근  print()  circle.set\_radius(3)  print("# 원의 둘레와 넓이를 구합니다.")  print("원의 둘레:", circle.get\_circumference())  print("원의 넓이:", circle.get\_area())  print()  class Circle:  def \_\_init\_\_(self, radius):  self.\_\_radius = radius  def get\_circumference(self):  return 2 \* math.pi \* self.\_\_radius  def get\_area(self):  return math.pi \* (self.\_\_radius \*\* 2)    @property  def radius(self): # getter  return self.\_\_radius  @radius.setter  def radius(self, value): # setter  self.\_\_radius = value   1. Pandas   # pandas를 활용하여 데이터 처리  import pandas as pd <- import로 pandas 불러오기  dict\_data = {'a':1, 'b':2, 'c':3} # 'a'는 key값  ar = pd.Series(dict\_data) # dictionary를 series 형식으로 변환  print(type(dict\_data))  print(type(ar))  print(dict\_data) # 딕셔너리  print(ar) # 시리즈  print(dict\_data['a'], " : " ,ar['a'],' : ', ar[0]) # series에서 'a'는 index명  print(ar[['a','c']])  ar[1:2] # index 첨자로 접근하면 마지막 첨자 위치 data 미포함★  ar['b':'c'] # index명으로 접근하면 마지막 data포함★  # => dictionary를 series로 변경 : dictionary의 key가 series의 index명이 됨  # 접근은 index명 또는 index 첨자로 접근 가능  dict\_data = {'a':[1,2,3], 'b':[4,5,6], 'c':[7,8,9]}  df = pd.DataFrame(dict\_data)  print('Print df :\n\n',df)  print()  # index명을 부여  df1 = pd.DataFrame(dict\_data, index=['idx1','idx2','idx3'])  print('Print df1 :\n\n',df1)  print()  a\_var = [[1,2,3],[4,5,6],[7,8,9]]  df2 = pd.DataFrame(a\_var,index=['idx1','idx2','idx3'],columns=['a1','b1','c1'])  print('Print df2 :\n\n',df2)  print()  print(df2.rename(index={'idx1':'ix1','idx2':'ix2','idx3':'ix3'}, inplace=True))  df2 |
| 오후 | 1. Index/Columns   행 index명 변경 : object.index = 새로운 행 index 배열  열 index명 변경 : object.columns = 새로운 열 index 배열  Ex)  a\_var = [[1,2,3],[4,5,6],[7,8,9]]  df = pd.DataFrame(a\_var)  df  df.index=['행1','행2','행3']  df.columns=['열1','열2','열3']  df  df1=df  print("df1 :",df1)  df1.rename(index={'행1':'ida','행2':'idb'}, inplace=True)  print(df1)   1. 행/열 삭제   : 행 삭제 axis=0, 열 삭제 axis=1  Ex)  df2 = df1  df2.drop('ida',axis=0, inplace=True) # axis=0 가로축부터 0으로 시작  print('df1 :',df1)  df2.drop('열1',axis=1, inplace=True) # axis=1 세로축부터 0으로 시작  print('df2 :',df2)   1. DataFrame 복사   df1 = df <- df의 주소를 복사, df의 값이 변하면 df1의 값도 변함  df2 = df[:] <- df의 값을 복사, df와 df2를 각기 다른 자료로 처리 가능  df1.drop('idb', axis=0, inplace=True) <- 원래 자료(df)가 바뀐다  # 문)  import pandas as pd  # DataFrame() 함수로 데이터프레임 변환, 변수 df에 저장  exam\_data = {'수학' : [90, 80, 70],'영어' : [98, 89, 95],\  '음악' : [85, 95, 100],'체육' : [100, 90, 90]}  df = pd.DataFrame(exam\_data, index=['서준','우현','인아'])  print(df)  print('\n')  # 데이터프레임 df를 복제하여 변수 df2에 저장, df2의 1개 행(row)삭제  df2 = df[:]  df2.drop('서준', inplace=True)  print(df2)  print()  # 데이터프레임 df를 복제하여 변수 df3에 저장. df3의 2개 행(row)삭제  df3 = df[:]  df3.drop(['서준','우현'], inplace=True)  print(df3)  print()  print(df)   1. 데이터 선택   #데이터 프레임의 행 데이터 선택 : loc, iloc  #행 선택 : loc , 정수형 위치 index : iloc  문)  exam\_data = {'수학' : [90, 80, 70],'영어' : [98, 89, 95],\  '음악' : [85, 95, 100],'체육' : [100, 90, 90]}  df = pd.DataFrame(exam\_data, index=['서준','우현','인아'])  print(df.loc['서준'])  print()  print(df.iloc[0])  print()  print(df.loc['서준':'인아']) # ['인아']자료도 포함  print()  print(df.iloc[0:2]) # 첨자 index2의 자료는 포함하지 않음  print()  print(df['수학']) # 컬럼 인덱스로 검색  print()  print(df.수학)  print()  print(df[['수학','영어']]) # 컬럼 인덱스로 검색  df.loc['일남']= [100,90,80,88]  # row로 데이터를 추가하는 경우 df.loc[index명] = [값...]  print(df)  #문 키보드에서 이름과 국어,영어,수학 점수를 입력받아 이름을 인덱스로 저장하는  # score dataframe을 생성  # 결과를 화면에 출력, 이름에 quit 입력되면 입력 종료  ->  def input\_score():  scores = input("국 영 수 점수 입력 >").split()  for i, score in enumerate(scores):  scores[i] = int(score) #데이터 타입 변환  return scores  import pandas as pd  df\_a =pd.DataFrame(columns=['국어','영어','수학']) # 컬럼명만 부여한 DF생성  while True:  name = input("이름 입력, quit -> 종료 :")  if name == 'quit':  break    score = input\_score()  df\_a.loc[name] = score # 입력된 name 인덱스로 점수 추가    df\_a  search\_name = input('검색할 이름 입력 :')  for name in list(df\_a.index):  if search\_name == name:  kor = df\_a.loc[search\_name,'국어']  eng = df\_a.loc[search\_name,'영어']  math = df\_a.loc[search\_name,'수학']  print("{} : {}, {}, {}, total = {}".format(search\_name,kor,eng,math,kor+eng+math))    문)  import pandas as pd  # DataFrame() 함수로 데이터프레임 변환. 변수 df에 저장  exam\_data = {'이름' : [ '서준', '우현', '인아'],  '수학' : [ 90, 80, 70],  '영어' : [ 98, 89, 95],  '음악' : [ 85, 95, 100],  '체육' : [ 100, 90, 90]}  df = pd.DataFrame(exam\_data)  # '이름' 열을 새로운 인덱스로 지정하고, df 객체에 변경사항 반영  df.set\_index('이름', inplace=True)  print(df)  print('\n')  # 데이터프레임 df의 특정 원소 1개 선택 ('서준'의 '음악' 점수)  a = df.loc['서준', '음악']  print(a)  b = df.iloc[0, 2]  print(b)  print('\n')  # 데이터프레임 df의 특정 원소 2개 이상 선택 ('서준'의 '음악', '체육' 점수)  c = df.loc['서준', ['음악', '체육']]  print(c)  d = df.iloc[0, [2, 3]]  print(d)  e = df.loc['서준', '음악':'체육'] # 음악부터 체육까지  print(e)  f = df.iloc[0, 2:]  print(f)  print('\n')  # df의 2개 이상의 행과 열로부터 원소 선택 ('서준', '우현'의 '음악', '체육' 점수)  g = df.loc[['서준', '우현'], ['음악', '체육']]  print(g)  h = df.iloc[[0, 1], [2, 3]]  print(h)  i = df.loc['서준':'우현', '음악':'체육']  print(i)  j = df.iloc[0:2, 2:]  print(j)  # 국어 점수를 모두 80점으로 추가  df['국어']=80  print(df)  df.loc["행추가"] = 0  df  # df의 2개 이상의 행과 열로부터 원소 선택 ('서준', '우현'의 '음악','체육' 점수)  print(df.loc[['서준','우현'],['음악','체육']]) |