<1. 인트로>

# 한 줄 주석이예요. (R의 주석과 같아요.)

# 여러 줄 주석은 """ 혹은 '''로 표현돼요

"""

이 안에 있는 건 다 주석이예요주석주석주석

python의 장점

- 상대적으로 쉬운 언어

- 강력한 데이터 분석 library (굉장히 많이 제공됨, 현재 가장 인기가 좋다!)

- Open Source

- R에 비해서 범용적인 사용이 가능

- 데이터 분석 결과를 다른 시스템과 연동시킬 때 좋다!

- 하위 호환성은 없어요~(2.x버전과 3.x버전이 달라요!)

- PyCharm 혹은 Jupyter notebook으로 작업해요.

"""

##############################################################

# Python의 Keyword

# Keyword module을 로딩해야 한다!

import keyword

# output영역에 출력할 때 사용하는 함수 : print()

print(keyword.kwlist)

# jupyter notebook hotkey(단축키)

# L : 라인번호 toggle

# O : output창을 toggle

# A : 위쪽 영역에 cell 삽입

# B : 아래쪽 영역에 cell 삽입

# DD : 현재 cell을 삭제

# Ctrl + Enter : 현재 cell 실행

# Alt + Enter : 현재 cell 실행하고 아래쪽에 cell 삽입

# Shift + Enter : 현재 cell을 실행하고 아래쪽 cell을 선택

## 변수를 생성하고 삭제해 보아요!

var1 = 100 # ;은 생략이 가능

var2 = 200

var2 # output에는 맨 마지막에 있는 변수명만 출력된다!! (여기선 var2 값만 출력됨)

# 출력할 때 print()를 이용하자

print(var1)

print(var2)

print("var2의 값은 : ",var2) # print 함수에는 인자를 여러개 줄 수 있다!

# 여러개의 인자를 하나씩 출력해요

#del var2; # var2를 메모리에서 삭제!

print("var2의 값은 : ",var2)

<2. 데이터타입>

# Python Built-in Type (파이썬의 내장 데이터 타입)

# Numeric(int(정수),float(실수),complex(복소수))

# Text Sequence : 문자열 (str)

# Bool : True, False (bool)

# 자료구조

# Sequence( list, tuple )

# Mapping( dict )

# Set( set )

# Numeric Data Type

# int (정수)

# float (실수)

# complex (복소수)

a = 123 # 정수(10진수)

b = 3.14592653 # 실수

c = 3.14E10 # 지수 표현

d = 1 + 2j # 복소수

e = 0o37 # 앞에 있는 0(숫자 0), 뒤에 있는 o(영문자) => 8진수

f = 0xFF # 16진수

print(type(a)) # a의 데이터타입을 알려줘요!

print(type(b))

print(type(c))

print(type(d))

print(type(e))

div = 3/4 # 0.75(버전 3.x에서 도출되는 값, 2.x버전: 0)

print(div)

result = 3\*\*3 # 지수표현

print(result)

result2 = 100 % 3 # 나머지 연산 (R에서는 %%)

print(result2)

result3 = 10//3 # 나눗셈의 몫

print(result3)

# Text Sequence

# 문자열 생성 방법 (4가지

a = "Hello" # ""를 이용

b = 'Hello' # ''를 이용

# 여러 줄짜리 문자열을 만들 때 사용

#c = """이것은

#소리 없는

#아우성"""

#d = '''이것은

#소리 없는

#아우성'''

# print(c)

# 문자열 연산. indexing, slicing

first = "이것은"

last = "소리없는 아우성!"

# 두 문자열을 연결

result = first + last # 연산자 overloading

print(result)

# 문자열의 곱

#text = "python"

#print(text \* 3) # 문자열을 3번 연결하는 것!

# a= "this is a sample text!!. show me the money!"

# indexing ( 0부터 시작 )

#print(a[0]) # t

#print(a[8]) # a

#print(a[-1]) # ! # -의 의미는 뒤부터 indexing!

# slicing ( 일부분을 발췌 )

# a = "this is a sample text!!. show me the money!"

#print(a[0:3]) # 시작은 inclusive, 마지막은 exclusive

#print(a[:4]) # 시작 부분이 없으면 처음부터(index 0 부터)

#print(a[4:]) # 마지막 부분이 없으면 끝까지!

#print(a[:]) # 문자열의 처음부터 끝까지!

# in, not in 연산자

a = "this is a Sample Text"

print("this" in a) # True ( TRUE X )

print("this" not in a) # False

print("sample" in a) # 대부분의 프로그래밍 언어는 대소문자 구분(html은 구분 안함)

print("sample" in a.lower()) # True

# 문자열 formatting

apple = 10;

a = "사과가 10개 있어요!!"

a = "사과가 " + str(apple) + "개 있어요!!"

print(a)

# 이렇게 할 수 있는데 너무 복잡스러워요!

# 이렇게도 포메팅 가능하다

a = "사과가 %d개 있어요!!" %apple

print(a)

a = "사과가 %f개 "

a = "사과가 %d개, 바나나가 %d개 있어요!!" %(apple,5)

print(a)

a = "원주율은 %f입니다." %3.141592

print(a)

a = "이것은 소리없는 %10s!!" %"아우성"

print(a) # 열칸 띄우고 오른쪽 정렬해라

a = "이것은 소리없는 %-10s!!" %"아우성"

print(a) # 열칸 띄우고 왼쪽 정렬해라

a = "원주율은 %0.4f" %3.141592 # 5번째 자리에서 반올림

print(a)

a = "원주율은 %10.4f" %3.141592 # 열칸 띄우고 5번째 자리에서 반올림

print(a)

## 문자열 함수

a = "cocacola"

# 문자열 길이

# result = len(a); print(result)

# result = a.count("co");print(result)

# result = a.find("o") ;print(result) # 찾는 글자가 있는 곳의 index 번호를 알려준다!

# a = ":"

# b = "abcd"

# result = a.join(b) ; print(result)

a = ["이것은","소리없는","아우성"]

b = " "

result = b.join(a) ;print(result) # 삽입할 문자를 앞에!!

print(type(result))

# a = " hoBBy "

# print(a.upper())

# print(a.lower())

# print(a)

# print(a.strip()) # 앞과 뒤의 공백을 제거

###############################################

# 출력양식을 설정할 때 formatting 이용

# apple =10

# a = "사과가 %d개 있어요!" %apple

# print(a)

a = "나는 사과를 {var1}개, 바나나 {var2}개 가지고 있어요!".format(var1=10,var2=5)

print(a)

# {} 안에 들어가는 숫자는 index번호!! 번호 안쓰면 순서대로 매핑됨!!

# {} 안에 변수명 넣고 뒤에서 변수 값 지정해줘도 된다!

# Numeric (정수, 실수), str

#########################################################

# 우리 자료구조에 대해서 알아보아요!

## Sequence Type : List

## list (파이썬의 가장 대표적인, 흔하게 사용되는 자료구조)

## 임의의 객체를 순차적으로 저장하는 자료구조

## 기호로는 [] (대괄호를 이용)

# myList = list() # 비어있는 list를 생성

# myList = [] # list를 literal로 표현

# myList = [1,2,3] # 3개의 값을 가지는 list를 생성

# myList = [1,3.14,True,"아우성!!"] # 모든 데이터타입을 다 저장할 수 있다!

# myList = [10,["Show","me","the","money"],3.14,True]

# # LIst안에 또 다른 자료구조들을 넣을 수 있어요!

# print(myList[2])

# print(myList[1])

# print(myList[1][3])

# print(myList[-1])

# print(myList[0:1]) # sclicing 결과는 원본자료구조와 동일하다(리스트 형태로 출력)

# # indexing과 slicing의 차이를 유념하자!

# print(myList[1][2:])

# # list의 연산

# a = [1,2,3]

# b = [4,5,6]

# print(a+b) # 두 개의 리스트가 이어지며 합쳐진다

# print(a\*3) # 리스트 원소가 n번 반복되며 리스트 출력

# # list의 indexing 과 slicing (기본적인 동작 방식)

# a = [1,2,3]

# #a[0] = 5 # [5,2,3]

# #print(a)

# #a[0:1] = [7,8,9] # [7,8,9,2,3] 슬라이싱은 영역

# #a[0] = [7,8,9] # [[7,8,9],2,3] 인덱싱은 위치

# del a[1]

# print(a)

# #a[1:] = [] # list slicing한 부분을 다른 리스트로 대체

# #print(a)

#list에 적용할 수 있는 함수

# a = [1,2,3]

# # a.append("Hello")

# print(a)

# #a.append([4,5,6]) # list의 맨 마지막칸을 생성, 그 안에 데이터를 저장해요!

# #print(a)

# a.extend([4,5,6]) # list를 확장(원소 이어지는 느낌)

# print(a)

# a = [7,3,9,4,6,1,8]

# # list를 정렬(오름차순, 내림차순)

# # result = a.sort() # a는 불변, result는 정렬된 결과? 아니다!!!

# # print(result)

# # 리턴 안되고 원본이 변경된다!!!

# # a.sort() # list가 가지고 있는 함수. 원본이 변경

# result = sorted(a) # python이 제공하는 함수 => 리턴 가능, 원본 불변

# print(result)

# print(a)

# a = ["Hello","kaka","World","haha"]

# a.sort() # 정렬의 기본(오름차순)

# # 문자일 경우, ASCII코드표를 기준으로

# print(a)

# a = ["아이유","김연아","백예린","새소년"]

# a.sort()

# print(a)

# a.reverse() # 리스트를 역순으로(뒤에서부터) 나열하는 함수

# print(a)

# b = ["홍길동","최길동","신사임당","아이유"]

# b.sort()

# print(b)

# b.sort(reverse=True)

# print(b)

# a = [1,2,3,4,5]

# # print(a.index(5)) # 찾은 다음 찾은 위치를 return

# # a.insert(0,"Hello") # 특정 위치에 값을 삽입할 수 있어요! (위치,넣을 값)

# # print(a)

# # a.remove(3) # 첫번째로 발견한 요소를 삭제(n번째는 삭제 안됨)

# # a.pop() # 맨 마지막 요소를 뜯어와요!(디폴트), 위치 잡아주면 해당 위치 값 뜯어옴

# print(a.pop(2))

# print(a)

a = [1,2,3,4,5,1,2,3,4]

print(a.count(1))# list안에 내가 찾으려는 요소가 몇개 있는지 return

# a = [1,2,3]

# b = [1,2,3]

# a == b # a와 b의 내용이 같은지를 물어보는 것!

# print(a == b) # True

# print(a is b) # False / a와 b가 동일한지 묻는 것! 둘은 완전히 같지 않다.

# a = [1,2,3]

# b = a # 내용뿐만 아니라 주소도 복사된 것

# print(a is b) # True

a = [1,2,3]

b = a[:] # b = [1,2,3]

print(b)

print(a is b) # False / b는 a의 값만 가져왔을 뿐 주소는 다르다!

print(id(a))

print(id(b))

# Sequence Type : tuple

# list와 거의 동일, but 표현하는 방식이 다르다!

# list는 대괄호로 표현 [1,2,3]

# tuple 소괄호로 표현 (1,2,3)

a = () # empty tuple

print(type(a))

myList = [1] # 요소가 1개인 list

print(type(myList))

myTuple = (1,) # 요소가 1개인 tuple을 만들때는 이렇게!

# (myTuple = (1)에 대해 출력하면 class가 int라고 나옴)

print(type(myTuple))

myTuple = (1,2,3) # Tuple은 list와 비슷해요 (Tuple 생성)

myTuple = 1,2,3 # Tuple은 () 생략 가능!

a,b,c = 10,20,30

(a,b,c) = (10,20,30)

print(a)

# 튜플은 함수 잘 안쓴다. 왜? 튜플 자체를 변경할 수 없어서~ (read only)

# print(myTuple[0:2]) # indexing & slicing 둘다 가능

# myTuple[0] =100 # 리스트와 달리, 튜플은 값을 바꿀 수 없다!!!

# # 이거 안된다! read only

# 리스트와 튜플

a = [1,2,3] # list

b = tuple(a) # tuple

print(b)

c = list(b) # list

print(c)

# Sequence type : range

# range1 = range(10)

# print(range1)

# range2 = range(1,11,2) # 파이썬은 시작은 포함, 끝은 불포함

# # [1,3,5,7,9] 형태로 인식

# print(range2)

# 숫자 영역을 잡는 range는 일반적으로 for구문을 사용할 때 많이 이용

# Sequence Type에는 결과적으로 3개가 존재

# list, tuple, range

# Mapping Type : dict (dictionary)

# key와 value의 쌍으로 표현되는 자료구조

# 예) JSON : ({name : "홍길동", age : 30})

# a = { "name" : "홍길동", "age" : 30 }

# print(type(a))

# # key 값은 불변의 값을 사용해야 해요!

# # list는 key로 사용할 수 없다! (값을 바꿀 수 있기 때문에)

# # tuple은 key로 사용할 수 있다!

# a[10] = "hohoho" # 이 문장만 보고서는 어떻게 쓰이는 구문인지 알 수 없음

# a["hobby"] = "Game" # 변수명[key값] = value

# print(a)

# print(a["age"])

# a = {"name" : "홍길동", "age" : 30, "age" :40} # 동일한 키 값이 있으면 안된다!!!

# print(a) # 두 개의 중복 키 값이 있으면 둘 중 하나에 문제 생김

# a = {"name" : "홍길동", ("address",) : "서울"}

# print(a)

# print(a[("address",)])

# # 튜플은 키 값으로 쓸 수 있다. 리스트는 못씀

#########################################################

# dict의 함수

# a = {"name" : "홍길동", "age" : 20, "address" : "서울"}

# value를 가져오고 싶다면? => key를 알아야해요

# a.keys() => # dict\_keys 자료구조로 리턴

# list와 유사한 자료구조

# myKey = a.keys()

# print(myKey)

# myList = list(myKey)

# print(myList)

# myValue = a.values() # dict\_values 형태로 가져와요

# print(myValue)

# myItem = a.items() # key & value를 리스트 형태로 만들어서 개별 튜플로 만들어줌

# print(myItem)

# dict안에 있는 모든 key와 value를 출력하세요! (for문 통해서)

# a = {"name" : "홍길동", "age" : 20, "address" : "서울"}

# for k in a.keys():

# print("{}-{}".format(k, a[k]))

# print("hoho")

# # \* python은 indent가지고 for문 블럭을 잡는다 \*

# print("age" in a) # 특정 key값이 dict안에 있는지 확인

## if문

area = ["seoul","busan","jeju"]

if "suwon" in area:

pass # 하는 일이 아무 것도 없을 때 사용

elif "seoul" in area:

print("서울이 있어요!")

else:

print("기타 등등")

for k in range(10):

print("k값은 : {}".format(k))

test = 1

while test < 10:

print("test값은 {}".format(test))

test += 1 # test = test + 1

a = 100

sum = 1000 # 절대 예약어로 변수 잡지 말 것!!!! 나중에 함수 쓸 때 처리X

a = [1,2,3]

sum(a)

## python

## 1. 환경설정

## - PyCharm IDE를 이용해서 개발

## 여러명이 같은 파일, 같은 프로젝트를 공유하면서 개발할 때 유리

## interactive한 개발은 힘들다!

## 데이터 분석 작업보다는 Web개발이나 다른 어플리케이션 작성

## - Jupyter Notebook

## interactive한 개발이 가능해서 데이터분석에 대한 학습이나 실제 분석작업을 많이 해요!

## 여러명이 공동 작업할 때는 불편~!

## 2. jupyter notebook 사용방법

## - cell 생성, cell 안의 코드 실행 방법, cell 삭제 방법

## - 무한루프가 되면 실행 interrupt 방법(위에 네모 버튼 누러면 중지됨)

## - 기본적인 fool 사용법

## 3. python built-in types

## - Numeric Type( int, float, complex )

## - Text Sequence Type( str ) => 좀 길게 설명

## - Sequence Type( list,tuple,range )

## - Mapping Type( dict )

## - Bool Type( bool ) : True, False를 사용해요

## - and : 논리연산자, or : 논리연산자, not : 논리연산자

## - & : 비교연산자. bitwise연산

## python에서 숫자 0은 False로 간주, 0이 아닌 모든 숫자는 True

## python에서 False는 숫자 0으로 변환, True를 숫자로 바꾸면 1로 변환

## 10진수 5를 2진수 => 0101

## 10진수 1을 2진수 => 0001

## True & False => 1 & 0 => 0001 & 0000 => 0000(마지막 자리는 서로 값이 달라서 0) => 0 => False

## python은 다음의 경우를 False로 간주

## : 숫자 0, 빈 문자열(""), 빈 리스트( [] ), 빈 튜플( () ), 빈 dict( {} ), None

print(bool("소리없는 아우성!!"))

print(bool([]))

print(bool(()))

print(bool({}))

print(bool(None))

print(int(True))

print(bool(100))

# result = True & False (X) => 파이썬에서는 & 또는 | 쓰지 말고 and 와 or 써준다!

result = True or False

print(result)

res = not True

print(res)

# 19-11-13 python의 남은 built-in type

# Set type( set )

# 어떻게 생겼나요? => {}

# {} : 기본적으로 dict. dict는 {}안에 key와 value로 구성

# set은 {}인에 요소만 들어감

# a = { 1, 2, 3 }

# b = {"name" : "홍길동", "age" : 20 } # dict

# print(type(a)) # dict 표현인데 key가 없다!

# print(type(b))

# ## set이라는 자료구는 기본적으로 순서가 없다. 중복을 허용하지 않는 저장장소

# a = set()

# a = set([1,2,3,4,5])

# print(a)

# print(type(a))

## 간단하게 set예제를 풀어보자!

#a = set([1,2,3,4,1,2,3,1,2,3]) # set은 중복을 배제해요!

#print(a)

#a = set("Hello") # 문자열을 set으로 만들면 어떻게 되나요?

#print(a)

# set의 연산

a = {1,2,3,4} # set의 literal (자료구조를)

b = {3,4,5,6}

# set => 집합 (집합연산, 교집합, 합집합, 차집합)

print(a & b) # & : 만약 set에 적용하면 교집합

print(a.intersection(b))

print(a|b) # | : 만약 set에 적용되면 합집합

print(a.union(b))

print(a-b) # - : 만약 set에 적용되면 차집합

print(a.difference(b))

mySet = {1,2,3,4,5}

mySet.add(7) # add()는 set에 원하는 요소 1개를 추가

print(mySet)

mySet.update([10,11,12]) # 여러개를 set에 추가

print(mySet)

mySet.remove(1) # 삭제

print(mySet)

print(len(a & b))

<3. 함수>

## function (함수)

a = [1,2,3,4]

# mySum = 0

# for i in a:

# mySum += i

# print(mySum)

mySum = sum(a) # python의 내장함수를 이용해서 처리

print(mySum)

## function (함수)

## 숫자를 입력받아서 => 단위기능으로 간주 => 함수

## 해당숫자를 모두 더한 후 => 단위기능으로 간주 => 함수

## "결과값은 : xx 입니다." 형식으로 출력 => 함수

## 이런 함수 기반의 프로그램 방식을

## 절차적 프로그래밍(procedural programming)

## python에서 함수는 크게 2가지로 구분

## 내장함수(sum()), 사용자 정의함수(user define 함수)

## 사용자 정의함수를 만드는 키워드는 def

## 함수 이름은 소문자로 시작(관용적으로)

## 함수의 정의(definition) => 함수의 이름, 인자형태, 인자개수 + 기능구현까지 모든걸 모은 것을 칭함

## 함수의 선언(declaration) => 함수의 이름, 인자형태, 인자개수까지만을 칭함

def mySum(a,b,c):

# 함수의 기능을 구현해요!

total = a + b + c

return total

inputValue = input() # 입력받기 위해 사용하는 내장함수 : input() => 무조건 str로 받아요!!

# imputValue = "100 200 300"

# 숫자 3개를 받아보아요!

print("입력받은 값은 : {}".format(inputValue))

var1 = inputValue.split(" ") # 리스트 형태로 결과 도출( 값을 분리해서 list로 return)

result = mySum(int(var1[0]),int(var1[1]),int(var1[2]))

print("최종 결과는 : {}".format(result))

## 2번째 함수

def mySum2(\*args): # args가 tuple로 간주됨

# args => (10,20,30,40)

# 전달받은 모든 값들을 더해서 리턴

k = 0

for i in range(len(args)):

k += args[i]

return k

total = mySum2(10,20,30,40)

print("최종결과 : {}".format(total))

## 함수를 하나 정의해요

## 숫자 2개 받아서 두개의 합과 두개의 곱을 리턴할거예요!

## tuple을 이용해서 마치 여러개의 값이 리턴되는 것처럼 사용할 수 있어요!

## tuple

# a = (10,20,30) # 오직 tuple만 소괄호 생략 가능!!!

# b = 10,20,30 # a와 b는 동치!!

def multi\_process(x,y):

result1 = x + y

result2 = x \* y

return result1,result2 # 값 두 개 리턴하는거 아니고 tuple하나를 리턴하는 것!!

a,b = multi\_process(100,200)

print(a)

## 사용자 정의함수를 만들어요!

## default parameter

def mySum(a,b,c=100): # 마지막 parameter에 default 값을 지정할 수 있다!

return a + b + c

total = mySum(1,2,3);print(total) # 6

total = mySum(1,2);print(total) # 103

## 변수의 scope

## (세상에는 여러명의 김철수가 있다. 하지만 어디에 있느냐에 따라 모두 다른 김철수)

tmp = 100 # 함수 밖에서 선언된 변수는 global variable( 전역변수 )

def myFunc(x): # 함수 안에서 선언된 변수는 local variable( 지역변수 )

# 지역변수는 함수가 호출될 때 생성되고 함수 수행이 끝나면 없어져요!

tmp = 10

# 만약 전역변수 tmp를 사용하고 싶다면 global tmp라고 써주면 됨

# global tmp (선언)

# tmp = 10 (지정)

tmp += x

return tmp

print(myFunc(20)) # 30

print(tmp) # 100

## 하지만 전역변수를 함수 안에서 써주면 함수가 외부와 유기적으로 연결돼버려서 좋지 않음

## 추천되지 않음. 그저 쓸테면 써보라고 기능 제공할뿐

## 사용하지 마세요!

## 사용자 정의함수는 위에서 설명한 방식대로 이용하면 된다.

## 프로그램을 작성할 때 함수를 어떻게 정의해서 사용할 것인가를 고민해서 작성

## 나머지는 내장함수예요. python이 제공해주는 함수

## int() : 인자로 들어온 놈을 정수로 변환

## abs() : 절대값을 리턴하는 함수

print(abs(-10))

## all() : 인자로 반복가능한 타입이 와야해요! 인자로 들어온 값이 모두 True일 경우 True 리턴

## 리스트, 셋, 튜플, 딕셔너리 등등이 가능 (int는 안됨)

a = [True, 100, {}, 3.14]

print(all(a))

## any() : 인자로 반복가능한 타입이 와야해요! 인자로 들어온 값 중 True가 있으면 True 리턴

print(any(a))

## len() : 길이 구하는 내장함수

## 튜플, 리스트 등등 대부분 사용 가능

## list() : 리스트 만들어주는 함수

## tuple() : 튜플로 변환하는 함수

## str() : 문자열로 변환하는 함수

## type() : 인자로 들어온 데이터의 type을 리턴

#@ sum(), max(), min() : 합계, 최대 최소를 구해줘요 (mean()은 없다!!)

a = [1,2,3,4,5]

print(sum(a)/len(a)) # mean 구하는 방법

## sorted() : 오름차순으로 정렬해주는 함수

a = ["이것은","소리없는","아우성"]

# "이것은 소리없는 아우성"

# join() 함수를 이용하면 편해요!

print("-".join(a))

################ 함수에 대한 기본 #######################

<4. 객체지향>

## 객체지향 프로그래밍(Object Oriented Programming)

## 함수기반의 프로그래밍을 하면 좋다!

## 1970년부터 시작해서 지금까지 이 방식으로 프로그래밍을 하고 있다

## 장점 : 프로그래밍이 쉬워요! => 빨리 만들 수 있어요!

## => 비용이 적게 들어요(인건비 줄어서)

## 단점 : 처음에는 단점이 별로 없어 보였다. 그러나,,,

## 1990년대 이후부터 세상이 변하기 시작했다. => 인터넷

## 정보량이 많아지기 시작 => 세상이 급변하기 시작

## 프로그램의 유지보수 요구가 많아지기 시작

## 절차적 프로그램으로 프로그램을 작성해서 이런 문제가 발생

## 어떤 방식으로 프로그램을 만들면 유지보수 측면에서 이점이 있을까?

## 객체지향 패러다임이 시작

## 객체지향이 어떤 프로그래밍 방식인지 파악

## 현실세계에 잘 부합하는

## 즉, 변화에 잘 적응하는 프로그램 만들기 위해

## 해결해야하는 문제를( 현실세계의 문제 ) 프로그램을 똑같이 모델링하면 되는거 아냐?

## 은행 프로그램을 만드려고 할 때

## 은행 프로그램을 구성하는 구성요소를 파악

## 행원, 고객, 계좌, 지점, 테이블, 돈,...

## 분석하는 사람마다 구성요소를 다르게 생각!

## 구성요소간의 행위들을 정의하는 프로그래밍 방식

## 예시) 은행계좌를 프로그램으로 표현( 객체 모델링 )

## 객체들은 크게 2가지로 파악할 수 있다

## 상태와 행위로 파악

## 상태는 값으로 표현되는 것, 행위는 동작으로 표현되는 것

## 잔액, 예금주, 계좌번호,... 등등 => 상태

## 출금, 입금, 조회 등등 => 행위

## ==> abstraction (추상화)

## class => 객체 모델링의 수단, 추상 데이터 타입

## 클래스의 이름은 대문자로 시작(관용적)

class Student:

### properties

# 상태(값) : 이름, 학과, 학년. 학번, 성별, 나이...키, 몸무게

# -> 변수로 표현 => properties

s\_nation = "" # class variable

### 생성자(constructor)

def \_\_init\_\_(self,n): # 고정된 형식

self.s\_name = n # instance variable( 객체가 독립적으로 가지는 저장공간)

self.s\_num = "" # instance variable

# 얘네는 class variable (모든 객체가 공유하는 변수)

self.s\_major = []

# Student.s\_name = n

### method

# 행위(동작) :

# => 함수로 표현

def display(self):

print("학생이름은 : {}".format(self.s\_name))

students = list()

students.append(Student("홍길동")) # 객체

students[0].display()

# k = Student("김연아") # 객체

# k.display()

# print(k.s\_nation)

# t = Student("홍길동") # 객체 생성

# # => 이 코드에 의해서 만들어진 메모리

# # 저장공간 (instance) => 객체

# # => 클래스 내부에 정의된 생성자라고 불리는 함수가 호출돼요

# t.display()

# Student.s\_nation = "한국"

# print(t.s\_nation)

<5. 파일처리>

## python의 기본 파일처리

## 먼저 파일 handle을 획득해야 해요!

## 사용할 데이터 파일을 오픈해요!

## 읽기용도, 쓰기용도

file1 = open("student\_score.txt","r")

file2 = open("backup.txt","w") # 하나의 파일의 내용을 다른 파일에 복붙하기 위한 용도로 만들어진 파일

# 학생 성적 파일에서 한줄씩 읽어와서

# output console에 출력하고 해당 내용을 백업 파일에 저장

while True:

line = file1.readline()

# print(line) # 출력한 다음 한줄을 띈다!

print(line, end = "") # 이렇게 써주면 줄 간격 없어진다

if not line: # 마지막 줄 끝나면 line 없는 상황이 되기 때문

break; # 가장 가까운 반복문을 탈출

file2.write(line)

file1.close()

file2.close()

# 아이유,92,94,99(개행문자 \n)

# 김연아,17,20,22(개행문자 \n)

print(type(line))

<6. 모듈>

## python은 module을 사용해요

## module : 하나의 python 파일 => 변수, 함수, class

## 파이썬의 module은 다른 파이썬 프로그램에서 불러와 사용할 수 있는

## 파이썬 파일을 지칭 (모든 파이썬 파일은 module로 사용할 수 있다!)

## 우리가 생성한 module을 어디에 저장해 놔야 할까요?

import sys # sys라는 module을 불러와서 사용할 거예요

sys.path.append("C:/python\_Lib") # sys.path가 리스트 형태인 것( 그래서 append 사용)

# 모듈을 저장할 수 있는 폴더를 지정 => C:/python\_Lib

## 주피터 노트북 킬 때마다 sys.path.append("C:/python\_Lib") 입력한 후 모듈 사용하는 것 보다는

## 환경변수 PYTHONPATH라는걸 만들어서 여기에 module을 넣어놓으면

## 프로그램을 어디에서나 사용할 수 있어요!

## module을 사용하는 형식은

## import 모듈명(파일명)

# import module1 # 확장자는 적지 않는다! 이름만!

# print(module1.module\_Pi)

# print(module1.mySum(10,20))

# alias를 이용해서 편하게 사용해보자!

# import module1 as m # (별명 붙여줘서 간단하게 사용하기)

# print(m.module\_Pi)

# print(m.mySum(10,20))

# from 모듈이름 import 모듈함수(or 변수 or 클래스)

# from module1 import module\_Pi

from module1 import \*

print(module\_Pi)

print(mySum(10,11))

# module과 package

# package : "." 기호를 이용해서 module을 계층적으로 관리

# package는 물리적으로 확인할 때는 => 폴더로 표현!

# 즉 패키지는 모듈을 관리하기 위한 폴더 개념인 거!

# A.B 라고 할 때 A : package, B : module

# import myPackage.module1

from myPackage import module1 # 패키지부터 명시하고 어떤 모듈 쓸지 정하고 사용,

# 앞으로 이런 형태 많이 사용할 것

print(module1.module\_Pi)

# from myPackage.module1 import mySum => 이렇게도 사용 가능

# print(mySum(10,20))

li=[]

for j in range(1,100):

t = 0

for i in range(1,j+1):

if j % i == 0:

t +=1

if t == 2:

li.append(j)

print(li)

<7. numpy>

## NumPy (Numerical Python)

## 수학적 계산을 할 때, 수치연산을 할 때 사용돼요.

## 마치 R의 Vector를 사용하는 것처럼 NumPy를 이용할 수 있어요!

## Vector연산, Matrix 연산을 빠르고 효율적으로 처리할 수 있어요

## Pandas, Matplotlib의 기본 모듈이 NumPy

## NumPy module은 ndarray(n-dimensional array)라고 불리는 자료구조를 이용해요 => R에서 Vector와 유사

## R의 백터 자료구조는 같은 데이터타입을 저장하는 자료구조

## ndarray도 같은 데이터 타입만 저장이 가능

## NumPy의 ndarray를 이용하려면 NumPy module이 설치되어 있어야 해요!

## (NumPy는 외부 모듈이기 때문!)

## module을 설치할 때 conda라는 명령을 이용해서 설치(아나콘다 사용할 경우에만 가능)

## python은 pip를 이용해서 module을 설치(아나콘다 환경이 아닐 경우)

## conda install numpy

import numpy as np

## ndarray를 생성하고 data type을 살펴보아요!

## python list

a = [1,2,3,4]

## 파이썬의 리스트는 원소 타입 상관없이 다 들어갈 수 있다는 장점이 있다 : 유연함

## 하지만 속도는 느리다는 단점이...

print(a)

print("list => {}, type => {}".format(a, type(a)))

print("list[0] => {}, type => {}".format(a[0], type(a[0])))

## 파이썬에서 데이터 타입을 알아보면 대부분 class로 잡혀있다

## NumPy ndarray

b = np.array([1,2,3,4])

print("ndarray = > {}".format(b))

print("ndarray type = > {}".format(type(b)))

print("ndarray dtype = > {}".format(b.dtype))

## int32 => 정수형이고 32bit로 구성되어 있어요!

## 2진수로 32개의 비트로 표현할 수 있는 최대 수는 => print(2\*\*32) = 4294967296

## 2진수 2개로 표현할 수 있는 수는 4 => 2\*\*2

## 2진수 32개로 표현할 수 있는 수는 ? => 2\*\*32

## int64를 이용하면 더 큰 수를 표현할 수 있어요!

print(2\*\*64)

## ndarray를 만드는 가장 간단한 방법 => numpy가 제공하는 array()함수 이용

c = np.array([100,"Hello",3.14])

print(c) # 모두 같은 데이터 타입이 되어야 해요!

print(c.dtype) # <U11 => 유니코드를 의미(=문자열)

print(type(c[0]))

## 넘파이일때 문자열과 그냥 퍼아썬에서 문자열은 값이 같아도 타입이 다르다...?

## 다차원 numpy array를 만들어 보아요!

myList = [[1,2,3],[4,5,6]] # 첫번째, 두번째 칸에 각각 튜플을 집어넣은 것

arr = np.array(myList) # 2차원의 ndarray

print(arr) # 2차원 형태의 NumPy array가 만들어졌다!

## 1행 2열의 값은 얼마인가요? 6

## 프로그래밍 언어이기 때문에 0행 0열부터 시작

print(arr[1,2]) # arr[1][2]처럼 쓰지 않아요!!

## index의 시작은 0부터 시작해요!!

## n차원의 ndarray가 어떻게 표현되는지 확인해야 해요!

## 기본 데이터타입은 정수일 경우 int32로 사용돼요!

print(arr.dtype)

## 만약 데이터타입을 지정하려면

arr = np.array(myList, dtype = "int64") # 그다지 좋지 않아요!

arr = np.array(myList, dtype = np.int64) # 이렇게 사용해요!!

arr = np.array(myList, dtype = np.float64)

print(arr.dtype)

print(arr)

# 튜플만 이용할 수 있나? 튜플도 이용할 수 있다! 일반 리스트를 넣어도 가능

## 차원의 개수와 크기 및 axis

myList = [1,2,3,4]

arr = np.array(myList) # python list를 이용해서 넘파이 array를 생성

## 이렇게 numpy array를 생성하면 차원의 개수를 알 수 있어요!

print(arr.ndim) # 차원의 개수를 알려주는 속성

print(arr)

## 차원의 개수와 원소 개수를 동시에 알려주는 속성

print(arr.shape) # (4,) => 1차원이고 원소의 개수가 4개 있다는 의미!

myList2 = [[1,2,3],[4,5,6]]

arr2 = np.array(myList2)

print(arr2)

print(arr2.shape)

myList3 = [[1,2],[3,4],[5,6],[7,8]]

arr3 = np.array(myList3)

print(arr3)

print(arr3.shape)

myList4 = [[[1,2],[3,4]],[[5,6],[7,8]]]

arr4 = np.array(myList4)

print(arr4.ndim) # 몇 차원인지 알려주는 속성

print(arr4)

print(arr4.shape)

## 1차원 => 열

## 2차원 => 행,열

## 3차원 => 면, 행, 열

# numpy array 어떻게 만드는지

# ndim이 어떤 속성인지

# shape은 어떤 속성인지

## 관련 속성 전부 기억해두자!

# numpy array 차원 제어 (numpy array가 가지는 특이한 점)

myList = [1,2,3,4]

arr = np.array(myList) # numpy array 생성(ndarray)

# 차원의 수를 알아보아요!

print("ndarray의 차원의 수는 : {}".format(arr.ndim))

## 차원의 수와 각 차원의 원소의 개수

print("ndarray의 shape : {}".format(arr.shape))

## ndarray의 크기를 알아보아요!! => 차원에 상관없이 모든 요소의 개수를 알아보기

print("ndarray의 요소개수 : {}".format(arr.size))

# python의 len()함수는 ndarray에 대해 적용하면

# 1차원의 요소개수를 리턴 (2차원 이상에서는 의미 달라짐)

print("ndarray의 len() : {}".format(len(arr)))

myList2 = [[1,2,3],[4,5,6]]

arr2 = np.array(myList2)

print("ndarray의 차원의 수는 : {}".format(arr2.ndim))

print("ndarray의 shape : {}".format(arr2.shape))

print("ndarray의 요소개수 : {}".format(arr2.size))

print("ndarray의 len() : {}".format(len(arr2))) # 1차원의 개수는 2개(리스트 2개로 이뤄져 있으니까)

myList3 = [1,2,3,4,5,6,7,8,9,10,11,12]

arr3 = np.array(myList3)

print(arr3)

print("ndarray의 차원의 수는 : {}".format(arr3.ndim))

## shape을 변경할 수 있어요!

# arr3.shape = (2,2,3)

arr3.shape = 2,1,2,3 # 튜플은 ()를 생략할 수 있다!

print("ndarray의 shape : {}".format(arr3.shape))

print(arr3)

print("ndarray의 요소개수 : {}".format(arr3.size))

print("ndarray의 len() : {}".format(len(arr3)))

## 이렇게 shape을 변경할 수 있는데

## arr.shape 처럼 shape을 직접 바꾸는 방식은 잘 사용되지 않아요!

## 다른 방식이 있어요! reshape() 함수를 이용해요!

# myLi = [1,2,3,4]

# ar = np.array(myLi)

# print(ar.reshape(2,2))

## numpy array의 type 변경

arr = np.array([1.1,3.14,2,44.0,3.99])

print("dtype : {}".format(arr.dtype))

print(arr)

## ndarray의 데이터 타입을 다른 형식으로 변경할 수 있어요!

# 실수를 정수로

int\_arr = arr.astype(np.int32)

print(int\_arr) # 실수를 정수로 변환할 때 소수점이하 버림처리

### numpy array를 생성하는 다양한 방법

arr = np.array([1,2,3,4]) # list를 이용해서 ndarray를 생성

## 모든 원소의 값이 0인 ndarray를 원하는 shape으로 생성

arr = np.zeros((3,4)) # (3,4) shape형태로 원소의 값이 0인 ndarray를 생성

# 튜플이 올 수 있지만 여기서는 ()생략 불가

print(arr)

## 모든 원소의 값이 1인 ndarray를 원하는 shape으로 생성

arr = np.ones((2,4), dtype = np.int32) # dtype을 명시해주면 해당 데이터타입으로 ndarray가 생성됨

print(arr)

## numpy array의 공간만 확보할 수 있어요! = 초기화를 하지 않아요!!

## 속도가 빨라짐! dtype은 들어오지 않는다.

arr = np.empty((5,7)) # 메모리 공간만 잡고 쓰레기 값이 들어온다..?

print(arr)

## ones나 zeros는 초기화 작업이 추가적으로 진행

## array의 크기가 커지면 속도가 저하.

arr = np.full((3,4),9.0) # 임의의 값으로 초기화를 진행

print(arr)

## 넌파이 어레이를 우리가 여러가지로 만들 수 있는데

## 그 때 활용되는 가장 기본 4가지 속성을 알아보았습니다.

## zeros, ones, empty, full

arr = np.array([(1,2,3),(4,5,6)]) # 2행3열짜리 ndarray

print(arr)

# 위에서 설명한 zeros, ones, empty, full와 같은 역할을 하면서

# shape이 지정한 array와 같도록 만드는 함수

arr\_like = np.ones\_like(arr, dtype = np.float64)

print(arr\_like)

## ndarray를 만드는 다양한 방법 => arrange

## python의 range()와 유사한 기능을 해요!

arr = np.arange(0,10,2) # 0은 포함, 10은 불포함, 2씩 증가 (시작, 끝, 증감치)

print(arr)

arr1 = np.arange(10) # 0부터 시작, 10은 불포함, 1씩 증가

print(arr1)

arr2 = np.arange(0.1,5.3) # 0.1부터 시작, 5.3 불포함, 1씩 증가

print(arr2)

## ndarray를 만드는 다양한 방법 => linspace

## np.linspace(start,stop,num)

## start부터 stop까지의 범위에서 num개를 균일한 간격으로 데이터를 생성하고

## ndarray로 return

## 눈으로 보기 힘드니 그래프로 보아요!

## chart를 그리는 대표적인 package가 있어요 => matplotlib

## conda install matplotlib -> 일단 설치

import matplotlib.pyplot as plt

arr = np.linspace(0,10,20) # 시작 값도, 끝 값도 포함

print(arr)

plt.plot(arr)

plt.show()

## NumPy Array(ndarray)

## 특징, 여러가지 속성들(ndim, shape...)

## ndarray 생성방법

## 난수를 이용해서 ndarray를 생성하는 방법

## 5가지 정도가 존재

## np.random.normal() : 난수를 생성해서 ndarray를 만들 수 있어요

## ==> 정규분포 확률밀도에서 난수를 발생시켜요!

import numpy as np

import matplotlib.pyplot as plt

## 평균

myMean = 50

## 표준편차

myStd = 2

arr = np.random.normal(myMean,myStd,(10000,)) # shape을 부여해야 shape대로 난수를 추출

# 일반 정규분포 형태로 추출할 것이기 때문에 평균과 표준편차도 부여해야 함

print(arr)

plt.hist(arr,bins=100)

plt.show()

# 랜덤값을 추출할때도 그냥 막 하는 것이 아니라 특정 분포에 따라 내용이 달라질 수 있기 때문에

# 생성 방법 알아둬야함

## 난수를 추출

## [0,1] 구간에서 균등분포 확률밀도상에서 난수 추출

## [0,1) : 0 이상 1 미만

## [0,1] : 0 이상 1 이하

arr = np.random.rand(1000) # 1차원

# (2,3)=> 2차원

# (1000,2,3)=> 3차원

plt.hist(arr,bins=100)

plt.show()

## 난수를 추출

## 표준정규분포 확률밀도 상에서 난수 추출

arr = np.random.randn(1000) # 인자의 개수로 차원을 지정

plt.hist(arr, bins=100)

plt.show()

## 난수를 추출(정수)

## 주어진 범위에서 균등분포 확률밀도 상에서 난수 추출

arr = np.random.randint(-100,100,(10000,))

plt.hist(arr, bins=100)

plt.show()

## 난수를 추출(실수를 추출)

## [0,1]범위에서 균등분포 확률밀도 상에서 난수 추출

arr= np.random.random((10000,)) #shape을 줘야 해요!!

plt.hist(arr, bins = 100)

plt.show()

## 난수는 랜덤값이기 때문에 실행할 때마다 추출되는 난수가 달라져야 해요!

# 난수값도 시실은 알고리즘에 의해서 계산되는 값이예요!!

## 알고리즘의 초기값을 고정시키면 항상 같은 난수를 얻을 수 있어요 => 재현성

np.random.seed(1000) # 초기값을 고정

# 동일한 seed 값에는 같은 원소만 추출, seed 번호 달라지면 다른 값 추출

arr = np.random.randint(0,10,(5,)) # 0부터 10사이, 1차원 5개

# 정수를 균등분포 추출

print(arr)

## 11월 19일

### NumPy (Numerical Python)

## (많은 양의 수학적인 연산을 빠르게 처리하기 위해 만든 모듈 : 수치 계산용 모듈 NumPy)

## NumPy 안에는 하나의 자료구조만 들어갈 수 있다 => ndarray

## array : 연속적으로 할당되고 같은 데이터 타입의 데이터를 저장할 수 있는 자료구조 (배열)

## : 인덱스를 이용해서 데이터를 저장할 수 있는 자료구조

## : R에서의 자료구조 array 때문에 혼동하지 말자! 1,2,...n차원 모두 가능

## 기본적으로 생성하는 방법

import numpy as np

arr = np.array([1,2,3,4])

## python에서 built-in type

## int, float, str, list, tuple, dict, set

## => 이 데이터 타입이라고 알고 있던 것들이 실제로는 몽땅 다 " class "로 되어있다.

## 예) <class int>

## type()이라는 함수로 class를 알아내요

print(type(arr[0]))

print(arr) # 리스트와 유사하지만 ,로 구분되어 나오지 않는다.

print(arr.ndim) # 차원의 수

print(arr.shape) # (4,) => 4개의 열로 4개의 요소가 있다

## 2차원 이상의 ndarray 만들기

myList = [[1,2,3],[4,5,6]]

arr = np.array(myList)

print(arr)

### 전체요소의 개수 알아내기

print(arr.size) # ndarray 안에 있는 모든 요소의 수

print(len(arr)) # 1차원의 요소 개수 (행 2개니까 2출력된다고 생각해도 된다)

## dtype : 원하는 형태로 요소 타입 변경

myList = [[1,2,3],[4,5,6]]

arr = np.array(myList, dtype = np.float64)

print(arr)

### 다양한 생성방법

### np.zeros(), np.ones(), np.full(), np.empty()

### np.arange(), np.linspace()

### random기반의 함수들 -> 4가지

## NumPy 활용에 대해 알아보아요!

## shape을 조절해서 내가 웒나느 형태의 ndarray를 만들어 보아요!

import numpy as np

arr = np.arange(0,12,1) # 1차원 형태의 넘파이 어레이 만들기

print(arr)

## 2차원으로 변경(4행 3열 2차원으로 변경)

## reshape을 이용하면 View가 만들어져요!!

arr1 = arr.reshape(4,3)

# arr1은 arr을 2차원데이터로 바라보는 창일뿐 => 원본이 바뀌면 arr1 이터도 바뀐다!

# reshape쓰면 모양이 바뀐 새로운 데이터가 생기는게 아니라 원본을 보여주는 창이 생김!

print(arr1)

arr[1]=100

print(arr)

print(arr1)

# view는 base속성이 있다!

print(arr1.base is arr) # arr이라는 넘파이어레이와 base가 같은지 묻는 것

## shape을 변경해서 새로운 ndarray를 만들고 싶어요!

arr = np.arange(0,12,1)

print(arr)

arr1 = arr.reshape(4,3).copy() # copy() 붙이면 새로운 ndarray를 생성 (view아님)

# arr1 = arr.reshape(5,3) # error (형태가 맞지 않아요)

arr1 = arr.reshape(2,-1) # 일단 2행으로 만들어! (앞에 있는거 채우고 남은거 맞춰서 열개수 생성)

print(arr1)

arr1 = arr.reshape(-1,3) # -1을 이용하면 계산하지 않아도 자동계산을 해줘요!

print(arr1)

arr = np.arange(0,100,1).reshape(5,-1).copy()

# 5행 20열짜리 넘파이 2차원 어레이가 된다

print(arr)

print(arr.ndim)

# 다차원배열을 1차원으로 만들 수 있어요

arr1 = np.ravel(arr) # ravel()은 한차원 낮춰주는 것?

print(arr1)

# 만약 2차원을 4차원으로 바꿔주라는 문제가 있다면

# 2차원을 1차원으로 바뀐 후 4차원으로 변경해주면 된다!!

## resize()는 reshape()과 유사

## reshape()은 형태가 맞지 않으면 변경시 error가 발생

## 하지만, resize()는 형태 안 맞아도 사용할 수 있다!!

arr = np.arange(0,10,1)

print(arr)

### resize를 이용하는 첫번째 방법

arr1 = np.resize(arr, (2,7)) # 변경된 ndarray를 리턴

print(arr1) # 모자란 부분은 배열의 처음부터 다시 추가

arr2 = np.resize(arr,(2,2)) # 남는 요소는 버려요!

print(arr2)

### resize를 이용하는 두번째 방법

# ndarray를 직접이용하는 방법

arr = np.arange(0,10,1)

result = arr.resize(2,3) # 원본이 바뀌고 return되지 않아요!

print(arr)

# 방법1은 원본은 바뀌지 않고 결과 값을 리턴받는 방식,

# 방법2는 원본을 직접 변경하는 방식

# (대부분 np.~~는 리턴을 받고, 원본.~~ 는 원본을 변경)

## NumPy array를 가로방향 또는 세로방향으로 결합

## 2행 3열짜리 ndarray를 생성

## 정수로 생성, 랜덤하게 균등분포로 0~10사이에서 추출(0부터 9까지)

## 6개를 추출해서 2차원 ndarray로 생성

np.random.seed(7) # 같은 값의 난수를 추출하게 고정시켜주는 속성

arr1 = np.random.randint(0,10,(2,3))

print(arr1)

## 가로방향으로 결합

## 행의 개수가 같아야 함! 열의 개수는 상관이 없어요~

arr2 = np.random.randint(0,10,(2,5));print(arr2)

result = np.hstack((arr1,arr2)) # \*주의\* tuple 인자 하나가 와야한다!!!

print(result) # 2행 8열짜리 데이터 생성

## 세로방향으로 결합

## 열의 개수가 같아야 함! 행의 개수는 상관이 없어요~

arr1 = np.random.randint(0,10,(2,3))

arr3 = np.random.randint(0,10,(3,3))

result = np.vstack((arr1,arr3))

print(result) # 5행 3열짜리 데이터 생성

#####################################################################################

# ndarray 생성 방법, 특징

# reshape => view개념!(메모리를 효율적으로 사용하기 위해서

# 구조는 그대로, 데이터를 보여주는 것. 원본에 요소 값을 변경하면

# reshape된 결과도 변경된다! resize는 아님!!)

# resize는 뷰가 아니고 새로운 데이터를 리턴해줘서 그 데이터를 array에 넣으면

# 새 데이터가 생성되는 것

# 여러개의 ndarray를 결합할 수 있어요 => hstack, vstack

# 이제 indexing과 slicing에 대한 얘기만 남았다!

## NumPy array의 indexing과 slicing

arr = np.arange(10,20,1)

print(arr)

for i in arr: # data만 추출해서 사용하는 for 문

print("data : {}".format(i))

for idx,data in enumerate(arr): # index번호가 필요한 경우엔 enumerate() 사용해주자!

print("index ; {}, data : {}".format(idx, data))

# 1차원에 대해서

arr = np.arange(0,5,1)

print(arr)

print(arr[2]) # indexing (list와 차이가 없어요)

print(arr[1:3]) # slicing [1 2](일정량의 데이터 뜯어오는 것)

# np array를 뜯어왔으니 결과값도 np array로 출력

print(arr[:3]) # [0 1 2]

print(arr[2:]) # [2 3 4]

print(arr[1:]) # [1 2 3 4]

print(arr[1::2]) # [1 3] # :2 <- 두칸씩 넘어가라는 의미

print(arr[-1:]) # 맨 마지막 요소만 슬라이싱 할 거예요

print(arr[-1]) # 이러면 indexing

# 2차원에 대해서

arr = np.arange(0,12,1).reshape(3,4).copy()

print(arr)

print(arr[1,2]) # 6 (indexing)

print(arr[:,0]) # [0 4 8] # : <- 전체

# 먼저 행으로 slicing한 다음, 열은 indexing 사용해서 추출

# 1차원 형태로 추출된다.

print(arr[:,0:1]) # 행과 열 모두 slicing 사용해서 추출

# 2차원 형태로 추출

# 인덱싱은 값으로 나옴. 슬라이싱은 영역으로 나옴

# 즉, 슬라이싱은 원본 데이터 형태 유지해서 출력

# [5 6 9 10]

print(arr[1:3, 1:3].ravel()) # 2차원 형태로 뽑힌 데이터를 1차원 형태로 출력하기 위해 ravel() 사용

## Boolean Indexing & Fancy Indexing

# 1. Boolean Indexing

## Boolean Indexing은 mask를 이용해서

## 배열의 각 요소의 선택여부를 True, False로 지정해서 True요소만 선택하는 방식

# 정수형 난수를 추출해보아요

# 0부터 9까지 정수형 난수를 1차원으로 10개 균등분포를 이용하여 추출

np.random.seed(1)

arr = np.random.randint(0,10,(10,))

print(arr)

## 여기에서 짝수만 추출할 것!

# 방법 1 : 로직을 이용해서 추출(추천 X)

result = list()

for i in arr:

if i % 2 ==0:

result.append(i)

result = np.array(result)

print(result)

# 방법 2 : Boolean Indexing

# 1. 조건에 맞는 mask를 만들어요!

# arr = [5 8 9 5 0 0 1 7 6 9]

# mask = [False True False False True True False False True False]

print(arr % 2 ==0) # 논리연산 (np array가 생성됨) # mask : arr % 2 ==0

print(arr[arr % 2 ==0]) # [8 0 0 6]

# 2. Fancy Indexing

## indexing을 할 때 배열을 이용해서 배열 요소를 찾는 방식

arr = np.arange(0,12,1).reshape(3,4).copy()

print(arr)

print(arr[2,2]) # 10

print(arr[1:2,2]) # [6]

print(arr[[0,2],2]) # [2 10]

print(arr[[0,2],1:]) # 2차원 np array가 출력됨

# [[ 1 2 3]

# [ 9 10 11]]

print(arr[:,[0,2]]) # [[ 0 2]

# [ 4 6]

# [ 8 10]]

print(arr[[1,2],[1,2]]) # ???조심! 행과 열 모두 Fancy indexing 안 된다!

# 단일 인덱스를 주거나 범위는 줄 수 있지만 행과 열 각각 Fancy는 불가능

# 위에처럼 한번에 추출은 불가능하지만 나눠서는 추출 가능

# 나눠서 추출하는 : 방법 1

print(arr[[1,2]][:,[1,2]])

# np.ix\_() 함수 사용해서 추출하는 : 방법 2

print(arr[np.ix\_([1,2],[1,2])])

## 기본적인 집계(통계)함수

# arr = np.arange(0,12,1).reshape(3,4).copy()

# print(arr)

# print(np.sum(arr)) # numpy가 가지고 있는 sum함수 이용

# print(arr.sum()) # arr이 가지고 있는 해당 클래스의 method 이용

# print(np.cumsum(arr)) # cumsum() <- 누적 함수 : 1차원으로 변환, 계속해서 누적시켜나간다!

# print(np.mean(arr)) # 평균

# print(np.max(arr)) # 최대(최소도 가능)

# print(np.std(arr)) # 표준편차

# print(np.sqrt(arr)) # (각 요소에 대한) 제곱근 구하기!

# print(np.log10(arr)) # 자연로그, 상용로그 사용할 수 있다!

## 축 개념을 이해해 보자

## 위에 있는 집계함수를 이용할 때 axis 개념을 사용

## 만약 axis를 명시하지 않으면 axis를 None으로 설정하고

## 대상범위를 전체 ndarray로 지정

# 1차원 ndarray

arr = np.arange(0,5,1)

print(arr)

print(arr.sum()) # axis를 명시하지 않은 경우

print(arr.sum(axis=0)) # 1차원에서 axis =0 이면 열방향 (1차원에서는 axis = 1 불가능)

# 2차원 ndarray

arr = np.arange(0,12,1).reshape(3,4).copy()

print(arr)

print(arr.sum(axis=0)) # 2차원에서 axis = 0 이면 행방향

print(arr.sum(axis=1)) # axis = 1 이면 열방향

# 3차원에서는 axis =0은 면단위, axis =1 이면 행단위, axis =2 이면 열단위

## 당.연.히. 일반 로직으로 프로그래밍하는 것보다 집계함수를 이용해서 처리하는게

## 훨~~씬 효율적!!

## 응용

np.random.seed(1)

arr = np.random.randint(1,10,(2,5))

# 5보다 큰 요소의 개수를 구하여라

(arr>5).sum() # mask 이용 : a > 5 값만 True로 떨어지고(=1) 그 빈도를 sum으로 더해줌

arr[arr>5].size # Fancy indexing

## delete() 함수는 axis을 기준으로 행과 열을 삭제

## 만약 축을 지정하지 않으면 1차배열로 변환한 후 해당 요소를 삭제

## 원본은 유지하고 삭제된 복사본을 return

arr = np.arange(0,12,1).reshape(3,4).copy()

print(arr)

# axis를 지정하지 않고 삭제 처리

result = np.delete(arr,1) # 1차 배열로 변환한 후 해당 인덱스를 삭제

print(result)

# 축을 지정

result = np.delete(arr,1,axis=0) # 1차 배열로 변환한 후 해당 인덱스를 삭제

print(result) # arr에서 2번째 행 삭제

result = np.delete(arr,1,axis=1) # 1차 배열로 변환한 후 해당 인덱스를 삭제

print(result) # arr에서 2번째 열 삭제