1. 시작

~~RTFM!!!~~

~~STFW!!!~~

1.1출처 및 참조 :

Jump 2 Python ★☆☆ #난이도

Byte of Python ★★☆

Scipy Lecture Notes ★★☆

<https://docs.python.org/3.5/library/index.html> # Python Standard Library *~~★★★~~*

<http://haerakai.tistory.com/category/Python/study>

<http://anandology.com/python-practice-book/getting-started.html>

1.2 설치 파일

Anaconda :

# 파이썬 2버전과 3버전을 바꿔가면서 쓰려면…

<https://conda.io/docs/py2or3.html#create-python-2-or-3-environments>

xming :

Kitty, Putty :

1.3 기본 설정

들여쓰기:

파이썬은 블록 설정이 매우 중요하며 잘못된 블럭 설정은 에러를 일으키기도 한다. 블럭을 새로 설정할 수 있는 조건은 if, for, 등 흐름 제어 파트에 나와있으니 참고. 블럭 구분을 위한 들여쓰기는 4칸이 적당하며 탭이나 vim의 >>,<<를 이용해야한다. .vimrc 파일에 set sw=4, set ts=4를 설정해 둘 것. 어떤 블럭은 세 칸으로 나뉘고 어떤 블럭은 네 칸으로 나뉘면 그것도 에러를 발생시키니 주의!

1.4 간단 팁

어떤 명령어의 도움말을 보고 싶으면 ‘명령어?’를 입력하면 되고 자세한 매뉴얼은 ‘man 명령어’를 입력하면 된다.

<tap>을 누르면 리눅스 터미널과 마찬가지로 그 뒤로 올 수 있는 명령어들이 팝업되므로 애용할 것. 도움말 찾아볼 일이 훨씬 줄어든다.

뒤에 더 자세히 나오지만, 함수() 이렇게 사용되는게 아니라 변수.함수() 형식으로 사용되는 명령어는 ‘Object Oriented Programming(객체 지향 프로그래밍)’이라 하여 프로그래머로 하여금 코딩을 훨씬 수월하게 하기 위해 파이선이 갖고있는 모듈, 클래스 등의 활용하는 과정에서 사용되는 문법이다. 뒤에서 자세히 볼 것.

2. 명령어, 내장함수

내장함수 이름으로 변수를 선언해둘 경우 해당 내장함수를 불러올 수 없으니 주의!

2.1 명령어, 메소드

print(obj\*,sep=’ ’,’end=’’) #default에서 sep은 띄어쓰기 1칸, end는 \n(줄넘김)

print(“it’s the {var}”.format(var=”answer”))

print(”I ate {0} apples. so I was sick for {1} days.”.format(number,day))

#0,1은 뒤에 나오는 포멧의 index

x.split(‘’) #‘’를 기준으로 나눈 리스트를 리턴.

x.replace(“”,””) #x안에 있는 “”를 “”로 대체.

x.strip() #양쪽 공백지우기. l- r-로 한쪽 공백만 지우는 것도 가능.

x.join(‘’) #x를 ‘’사이에 넣음

x.upper() #or .lower

x.index(‘’) #’’를 찾아서 위치 리턴. 없으면 오류 발생

x.find(‘’) #위와 같으나 없으면 -1

x.count(‘’) #’’가 들어있는 개수 리턴.

x.startswith(‘’) #’’로 시작하는 x일 경우 true

#python 2.7의 경우 괄호없이 print x 로 쓴다.

2.2 논리형 내장함수

all(iterable) #iterable이 모두 참이면(맞으면) True, 하나라도 거짓이면 False를 리턴.

any(iterable) #iterable 중에 하나라도 참이면 True, 하나라도 거짓이면 False를 리턴.

isinstance(object,class) #object가 class에 해당되면 True, 아니면 False를 리턴.

2.3 Iterable 처리 함수

filter(function,iterable) #iterable을 차례로 불러와 function의 결과가 True인 값만   
 filter 객체로 리턴. list(filter())을 사용하면 리스트로 출력가능.  
list(iterable) #리스트이든 아니든 iterable한 object를 리스트로 리턴.  
tuple(iterable) #위와 비슷, 튜플로 리턴.

map(function,iterable) #function이 차례로 수행된 iterable들이 map class로 리턴.   
 list(map())로 쓰면 리스트로 출력됨.

max(iterable) #생략. 알파벳도 된다. 숫자와 알파벳이 섞여있으면 에러.  
min(iterable) #생략. 알파벳도 된다.  
zip(iterable\*) #여러개의 데이터를 하나씩 튜플로 묶어서 zip class로 리턴.   
 list(zip())으로 하면 튜플들로 이루어진 리스트로 리턴됨.

2.4 lambda

함수를 변수처럼 간단하게 다루게 해주는 명령어. 문법은 lambda args\* : 인수들로 이루어진 식 이며, function class의 object를 리턴. object이기 때문에 리스트에 넣는 등 일반적인 함수 선언인   
def function: 로는 사용할 수 없는 방법으로 활용이 가능하다.

lambda에는 변수 없이 그냥 식만 넣어도 돌아간다.  
lambda:”{} World”.format(“Hello”)

2.5 기타 내장함수

ord(str) #chr()은 해당 숫자의 아스키코드 값을 문자로 리턴하는 함수.   
chr(value) #ord()는 반대로 아스키코드(숫자)를 리턴하는 함수  
input(“str”) #{str}이 출력되며 입력값을 받음. 보통 var=input(“str”)으로 쓰임.  
int(var) #값을 정수로 출력.

3. 개체

4.1 문자열

x='abcd', y=’efgh’ #배열로 취급, 원소 변경 불가.

“=”\*2 #==

x+y #’abcdefgh’

x=pithon

x=x[0]+”y”+x[2:] #”python” 문자열은 그 안에서 변경이 불가능하니 새로 만들어줘야함.

str.[tap] 참조.

[문자열 포맷 코드] #.format 쓰면 되서 거의 쓸일 없을듯.

코드 설명

%s 문자열 (String)

%c 문자 1개(character)

%d 정수 (Integer)

%f 부동소수 (floating-point)

%o 8진수

%x 16진수

%% Literal % (문자 % 자체)

4.2 목록 list

x=[1,2,3,4] #4개짜리 리스트

x.sort() #크기 순/알파벳 순으로 정렬,

x.append(i) #원소 i를 뒤에 추가

x.reverse() #순서를 역순으로.

x.index([]) #[]의 값의 위치를 리턴. 없으면 에러.

x.insert(a,b) #index a에 b라는 값을 넣는 함수.

x.remove([]) #[]를 제거. 단, 중복되는 값이 있다면 첫번째로 나오는 []에만 적용.

x.pop() #x의 제일 뒤 항을 출력 뒤 그 항을 삭제. 숫자를 넣으면 해당 index의 값이 출력뒤 삭제.

x.count() #개수세줌.

x.extend([]) #리스트에 []를 더함.(x+[]와 같음)

del x[i] #i번째 원소 삭제

x=[1,2,3,[4,5,6],7,8,9] #원소안에 들어가 있는 리스트, x[3][2]를 불러오면 0,1,2,3... 에서 [4,5,6]이 출력,   
 #그뒤에 [2]를 붙였기 때문에 [4,5,6]의 세번째 원소인 6이 출력된다.

y=[i\*\*2 for i in x] #x의 제곱의 list 생성.

z=[m-n for (m,n) in zip(x,y) #x에서 y를 뺀 list 생성

4.3 튜플 Tuple

c, d=d, c #두 변수의 값을 바꾸는 식.

#배열이나 스칼라로는 복잡한 문제지만 간단하게 한줄로 해결된다.

x=() #원소가 없는 튜플

x=1, #원소가 한 개인 튜플

tup[0]=x #에러발생, 튜플은 원소 하나하나에 대해 변경 불가능

4.4 딕셔너리 Dictionary

기본적인 형태는 {Key1:Val1, Key2:Val2, ...}이다. 중괄호로 묶여있다. dic= 으로 선언, key와 value를 추가하려면 dic[KeyN]=ValN  
리스트와 비슷하게, dic[KeyN]을 입력하면 ValN이 리턴된다. (리스트가 lis[N]=ValN이었던 것처럼)

Key들로 이루어진 리스트를 만들려면 dic.keys()라는 함수를 이용, dict\_keys라는 object를 리턴받을 수 있다. 이외에도 .values ( [val\*] ) .items ( [(key,val)\*] )도 사용가능하며, list()를 사용하면 리스트로 만들어 사용할 수도 있다.

del dic[Key] #Key,Val을 삭제

Key in dic #Key가 있으면 True, 없으면 False.

\*Dictionary 사용시 주의할 점:

1. Key값이 중복될 경우 하나의 Key만 사용가능하고 나머지는 무시된다.

2. Key값에는 변할 수 있는 값은 넣을 수 없음. (ex. 리스트) 따라서 tuple, str는 가능.

4.5 집합형 자료 (Set)

문법은 S = set([ , , , , , ... ])이며, 중복되는 값을 추가하거나 이더레이터를 넣었을 때 중복값들이 하나만 저장되고 인덱싱이 없어지는 것이 특징이다. 교집합, 합집합 등 집합으로 처리할 때 사용하면 유용함.

4.6 기타

dtype(data type, 문법상에서 dtype=’’ 등으로 쓰임)

bool(True or False), float(실수), int(정수), 등등

4. 흐름제어

3.2 if

if x==y : # : 꼭 붙여야 함!

() #실행문 앞에서는 tap 한 번 꼭 해줄 것!!

elif x!=z: if 조건 밑에 실행문은 하나의 블럭으로 구분해주어야한다.

()

else:

() # else if -> elif, 같지 않음 -> !=

3.3 Iterable, Iterator, Generator, Function

Iteration은 반복이라는 뜻으로, 순서가 존재해 for문이나 .next()(or next(), python 3)를 이용해 요소들을 순서대로 불러 오는 일을 말하며, iterable은 iteration이 가능하다는 뜻이다. 특이하게도, 리스트는 iterable하지만 iterator는 아니기 때문에 {list}.next()(or next({list}), python 3)를 실행하면 오류가 발생한다.

iterable은 리스트 뿐만 아니라 파일(1줄씩 iteration), string(한 글자씩), Dictionary(key 하나씩)도 가능하며, iterator로 활용하고 싶으면 iter({})를 쓰면 된다.

#iterator 활용 예

list(iter), sum(iter), ...

var=iter(iterable) #iterator 생성, iterable로부터 요소들을 불러와 iterator를 만듦.

Generator, 즉 생성기는 return 대신 yield 명령어가 포함된 함수를 지칭한다. 이때 yield란 값을 반환하되 함수의 진행상황을 그대로 유지하라는 명령어로, 이를 return으로 바꾸면 일반적인 함수와 같다. 진행상황을 유지한다는 점에서 for문(반복해서 함수를 불러오는 상황에 유용)과 주로 자주 쓰이며, iterator를 생성, 활용하는 코드를 짤때 일반 함수를 사용하는 것보다 훨씬 효율적으로 활용이 가능하다.

({var를 포함한 식} for var in {iterable}) #generator expression, 간단한 제너레이터 생성.

value\_generator = (very\_slow\_function(x) for x in xrange(1, 100))

for value in value\_generator:

print(value); break #generator가 아니라 list로 [f(x) for x in xrange(1,100)]이었다면

생성할때 1부터 99까지 다 계산을 해야하지만 generator로 선언했기 때문에

x=1일때 계산하고 2일때 계산하고 그때 그때 처리되서 불필요한 계산을

하지 않을 수 있고 더 큰 계산에서는 메모리도 훨씬 적게 차지하게 할 수 있다.

제너레이터에서 변수 하나씩 끌어다 쓰는 방법은 next(gen)이며, 제너레이터를 활용해서 끝까지 돌리고 나면 읽었던 곳부터 시작하기 때문에 다음에 다시 쓰려면 아무것도 못 가져온다. 처음부터 다시 시작하려면 제너레이터 선언을 새로 해야한다. 빠르게 하려면 itertools의 tee() 함수를 불러와서 gen, bupgen = tee(gen) 꼴로 만들어서 사용하면 원래꺼 한번, 백업된거 bupgen 한번 두번 쓸 수 있게 만들 수 있긴하다.

3.3 for

for x in [1,2,3,4] #list의 각각의 x에 대해 반복, 지역변수 x를 선언, 사용 가능

y=[x\*\*2 for x in [1,2,3,4]] #list comprehension

y=[x\*\*2 for x in [1,2,3,4] if x%2 == 0]

for i in args:

total += pow(i,power) #y +=i는 y에다 i를 계속 더하라는 뜻.  
 #(y가 0일때) y+=1 -> 1,2,3,4, ...

(+) 내장함수 enumerate

for index, value in enumerate({iterable}):

print(index, value)

0 {0}

1 {1}

2 {2} #리스트의 각 항과 인덱스로 이루어진 튜플 iterable 세트를 리턴하는 함수.   
 for와 같이 사용하는 경우가 많음.

3.4 zip

iteratable한 데이터 둘을 묶어 zip형식의 객체로 리턴하는 명령어. 병렬로 이더레이터를 진행할때 유용하게 활용 가능하다. 예를 들어,

name=[‘Pig’,’Tiger’,’Dinorsour’]

letter=[len(n) for n in name]

이라는 자료에서 name의 길이 중 가장 긴 값을 찾으려한다면, name(i)를 하나씩 불러오는 이더레이션과 불러온 name(i)로부터 글자의 길이를 확인하는 letter의 이더레이션이 병렬하여 진행되는데 이때 zip을 이용하면,

for name,count in zip(names,letters): #count = letter[index]와 같은 처리가 줄어든다.

if count > max\_letters:

longest\_name = name  
 max\_letters = count

단, 이 때 zip(A,B)에서 A,B의 길이가 다르면 가장 짧은 걸로 작동하기 때문에 사용에 유의할 것.

파일관리

.1파일 입출력 명령어

f=open(“name”,’\*\*’) #r 읽기, w 쓰기, a+ 덧붙이기, b 바이너리, python3같은 경우

. rb,wb로 써야한다는데 자세한 것은 이후 추가.

<Read>

data=f.\*\* #read():파일 전체 바이너리 한 줄로 읽기. \n같은거 포함

#readlines():전체 파일 읽어서 리스트 안에 한 줄마다 한 요소씩 배치

#readline(size=-1):한 줄씩 읽기, 값 넣으면 그 글자 수 만큼만 읽음.

np.loadtxt() #txt파일 읽을때 쓰면 좋음

np.genfromtxt() # csv(comma-separated values) 읽을 때 쓰면 좋음

np.load() #np파일이나 binary format 읽을 때 씀.

<Write>

쓰기 시에 주의할 점 : 파일이 없으면 새로 만들고, 있으면 원래 파일은 없어진다. 원래 있던 파일을 손보는 경우 ‘a’를 쓸 것!

f.write(data)

f.close()

np.savetxt()

np.recfromcsv() # csv(comma-separated values) 쓸 때 쓰면 좋음

np.save() #np파일이나 binary format 으로 작성할 때 씀.

(+) with 구문 활용

with open(“name”,’\*\*’) as f:

f.write(\*\*) #이렇게 사용하면 f.close()를 사용하지 않아도 됨. f.open을 대신 함.

.2 pickle

복잡한 자료를 파일에 읽고 쓸수 있게 해주는 모듈. 몹시 놀라운 기능이라고한다...? 나는 아직 잘 모르겠지만.

<문법>

import pickle

.dump(var,dir)

.load()

.loads()

.3 glob

파일들의 목록을 추려낸다.

<문법>

import glob

glob.glob(‘\*\*’) #이름.확장자 ex)glob.glob(‘\*.\*’)#모든 확장자, 파일 출력

.4 hdf5

hdf5( Hierarchical Data Format) 많은 양에 데이터를 체계적으로 분류, 저장할 수 있는 파일포멧. 위성자료 받으면 이거인 경우가 많음. 파이썬에서는 직접적으로 제공하지는 않으나 역시나 갓모듈이 잘 지원하고 있음.

<문법>

import h5py

f = h5py.File(“filename.h5”, ‘r+,r,w,a’) #오픈 형식은 물론 하나만 적는거임

gr = f.create\_group(‘grname’) #생성 위치는 root group

ds = f.create\_dataset(‘dsname’) #생성 위치는 rootgroup

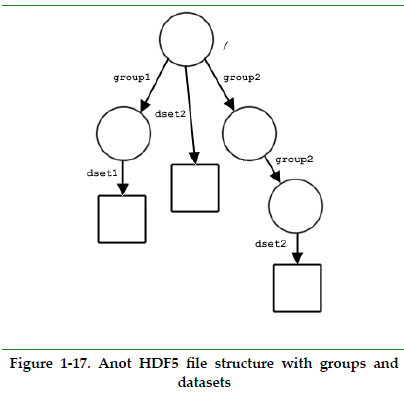
sgr = gr.create\_group(“grname2”) #생성 위치는 gr(위에서 선언한)

sgr = f.create\_group(“/grname/grname2”) #위와 같은 결과. 절대경로를 이용한 경우.

rdds = f.[‘/dsname’] #생성된 데이터셋 읽기, 보다시피 리눅스랑 비슷하다.

list(f) #root group에 있는 컨텐츠를 볼 수 있다.

list(f.keys) #root group에 있는 컨텐츠를 볼 수 있다. ~~이거 찾는데 한 시간 넘게 걸렸다~~.

<구조>

f – 파일(File)

: 불러온 파일, 이것 저것 들어있지만 볼려면 하나씩 다 까봐야된다.

ds – 데이터셋(Dataset)

: 데이터가 들어있는 공간. 그룹이 디렉토리라면, 데이터셋은 파일이다.

dt – 데이터(Data)

: 데이터 셋 안에 들어있는 값들. 별 의미는 없다. 그냥 임의 선언하고 사용하는, 말 그대로 데이터.

gr – 그룹(Group)

: 터미널의 디렉토리같은 개념으로, 폴더처럼 만들어서 사용하는 공간. 값을 지정할 수는 없으며, 대신 안에 데이터 셋을 넣어서 따로 보관, 꺼내올 수 있다. 파이썬 제공 객체인 딕셔너리처럼 작용하여 gr.keys(), gr.values() 처럼 사용하여 list로 읽어 올 수 있는데, 자세한 건 다른 항목(Dictionary)을 참조하자.

.5 .xdr

idl에서 주로 사용되는 데이터 저장 포멧으로, xdr reading으로 검색해서 나오는 모듈로 처리하려면 굉장히 불편하게 사용해야하지만 idlsave모듈을 사용하면 매우 간편하게 사용할 수 있다. 자세한건 help() 이용해서 찾아볼 것. 현재 scipy로 넘어가서 idlsave라는 모듈은 업데이트가 안되고 있지만 scipy로 넘어가있는 모듈 from scipy.io.idl import readsav 과 비교해도 불편한 점은 없으니 취향에 따라 골라 쓸 것.

<문법>

import idlsave

s = idlsave.read(‘filename.xdr’)

혹은,

from scipy.io.idl import readsav

readsav(‘filename.xdr’)

<FTP(File Transfer Protocol)>

네트워크에서 컴퓨터끼리 데이터를 교환하는 목적으로 만들어진 통신 규약. 데이터 교환이 목적이기 때문에 안정성이 매우 높다고...~~하지만 사실 코드 짜놓으면 다운로드하기가 몹시 간편하기 때문에 사용하는 듯...~~

간단히 코드 짜놓은 걸 하나씩 이해해보자. 스페인어이긴 하지만...

# coding=utf-8

from ftplib import FTP # Importamos la libreria ftplib desde FTP

import sys

def imprimirMensaje(): # Definimos la funcion para Imprimir el mensaje de bienvenida

print "------------------------------------------------------"

print "-- COMMAND LINE EXAMPLE --"

print "------------------------------------------------------"

print ""

print ">>> Cliente FTP en Python "

print ""

print ">>> python <appname>.py <host> <port> <user> <pass> "

print "------------------------------------------------------"

def f(s): # Funcion para imprimir por pantalla los datos

print s

def download(j): # Funcion para descargarnos el fichero que indiquemos según numero

print "Descargando=>",files[j]

fhandle = open(files[j], 'wb')

ftp.retrbinary('RETR ' + files[j], fhandle.write) # Imprimimos por pantalla lo que estamos descargando #fhandle.close()

fhandle.close()

ip = sys.argv[1] # Recogemos la IP desde la linea de comandos sys.argv[1]

puerto = sys.argv[2] # Recogemos el PUERTO desde la linea de comandos sys.argv[2]

usuario = sys.argv[3] # Recogemos el USUARIO desde la linea de comandos sys.argv[3]

password = sys.argv[4] # Recogemos el PASSWORD desde la linea de comandos sys.argv[4]

ftp = FTP(ip) # Creamos un objeto realizando una instancia de FTP pasandole la IP

ftp.login(usuario,password) # Asignamos al objeto ftp el usuario y la contraseña

files = ftp.nlst() # Ponemos en una lista los directorios obtenidos del FTP

for i,v in enumerate(files,1): # Imprimimos por pantalla el listado de directorios enumerados

print i,"->",v

print ""

i = int(raw\_input("Pon un Nº para descargar el archivo or pulsa 0 para descargarlos\n")) # Introducimos algun numero para descargar el fichero que queramos. Lo convertimos en integer

if i==0: # Si elegimos el valor 0 nos decargamos todos los ficheros del directorio

for j in range(len(files)): # Hacemos un for para la lista files y

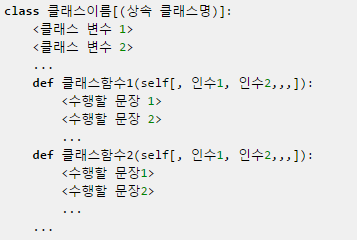
download(j) # llamamos a la funcion download para descargar los ficheros

if i>0 and i<=len(files): # Si elegimos unicamente un numero para descargarnos el elemento nos lo descargamos. Comprobamos que sea mayor de 0 y menor que la longitud de files

download(i-1)

5. 객체 지향 프로그래밍

5.1 클래스(Class), 객체(Object)



Class

임의로 지정할 수도 있는 객체의 형식. 값을 입력하면 함수를 적용하여 출력한다는 점에서 함수 여러개를 묶어 놓은 것과 비슷. 다른 객체들과 같은 틀 안에서 사용 가능하게 만듦. 이때 객체 하나하나를 클래스의 인스턴스라고 볼 수 있다. 다음을 인용하여 설명을 덧붙여본다.

“객체는 클래스의 타입으로 선언되었을 때를 의미하는 것이고,   
 그 객체가 메모리에 할당되어 실제 사용될 때를 인스턴스라고 하는 것이다.

Method

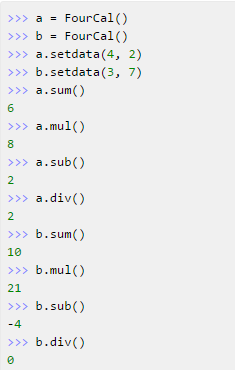
이때 클래스 안에서의 함수를 메서드(Method)라고 하며, func이라고 할때, A.func()로 사용이 가능하며, class안에 정의된 대로 실행됨.

Self

이때 함수가 func(self,arg\* ) 으로 정의되었다면, 이때 self는 클래스를 불러온 객체 자신을 의미한다. 이게 함수와 매서드가 다른 한가지다. 나머지는 함수와 같음. 예를 들어, 부를 때의 형태가 A.func(arg1,arg2)이이라면, self를 생략해도 A가 들어가서 매서드로서는 func(A.func(arg1,arg2) 형태로 작동된다. 클래스 안에서 메서드를 정의할 때는 아무런 인수가 없는 함수라고 해도 self는 적어줘야한다.

객체 변수

여기서 주목해야할 것은 클래스의 변수 arg1,2,3,....가 객체별로 유지될 수 있다는 점. A.arg1=[]형태로 쓰면 arg1에 값이 지정될 수 있고, 이때 arg1를 객체변수라고 부름. 이렇게 저장된 객체변수는 print(A.arg1)처럼 다른 명령때 활용이 가능하다. 객체변수 지정은 클래스 안에서 계산 될 수도, 직접 지정, 입력할 수도 있다.



# 함수 x.setdata(a,b)는 x의 객체변수 두 개(self.arg1=a, self.arg2=b)를 지정하는 함수.

FourCal()에서 지정한 sum, sub, div, mul 함수는 변수 self.arg1, self.arg2에 대해 계산하는 함수.

# a.arg1=5를 입력하고 a.sum()을 실행하면 7이 출력됨.

클래스 변수 ( A.var, B.var, C.var ...)

그 클래스로부터 생성된 모든 인스턴스 들이 접근할 수 있는 변수. 각각의 개체에 대해 값을 갖는 객체 변수와 대조적으로 모든 인스턴스들이 공유하며, 객체로부터, 혹은 클래스 자체를 바꿈으로써 변수가 바뀌면 모든 인스턴스들에 변경사항이 적용된다.

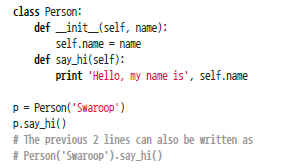
\_\_init\_\_ Method

클래스가 인스턴스화 될 때 자동으로 초기화처럼 실행되는 메소드. def \_\_init\_\_(self, arg) 등으로 사용되며, 모든 인스턴스에 대해 생성단계에서 적용된다는 점에서 용도에 따라 간편하게 만들 수 있음.

Class에 \_\_init\_\_이 없었다면,

p=Person()  
p.name=’Swaroop’  
p.say\_hi()

이렇게 적어야 같은 결과를 볼 수 있음.



\_\_str\_\_ Method

클래스를 프린트했을 때 출력하는 값을 정하는 메소드. 다음과 같이 활용.

Func(a,b,c)

def \_\_str\_\_(self): #print(func(a,b,c))를 실행하면 a: <b,c>를 출력

return self.a + ': <' + str(self.b)\

+', ' + str(self.b) + '>'

#클래스를 선언할 때는 이 두가지 메서드를 선언해두는 습관을 갖는게 나중에 디버깅할때나 코드를 이해할 때 편하다.

상속

클래스는 서로 종속관계에 둘 수 있음. 문법은 subclass(superclass), subclass1(superclass),...이며 하나의 클래스 안의 여러개의 서브 클래스를 만들어 상속시킬 수 있다.

이 문법의 특징은 상위 클래스의 속성이 하위 클래스의 그대로 상속되고 하위 클래스는 상위 클래스의 함수, 변수들을 유지한채 다른 식을 추가한다는 점인데, 이로 인해 클래스 간 공통된 함수, 변수를 수정할 필요가 있을때 상위 클래스만 손보고 나머지는 상속에 의해 수정되도록 하여 간편하게 코드를 짤 수 있다.

반면, 하위 클래스에서 기존의 매서드(함수)를 동일한 이름으로 다르게 설정하면, 하위 매서드로 덮어써서(오버라이딩) 다른 함수로 활용된다.

또, 두개의 객체에 대해 연산자를 활용하여 함수를 선언할 수도 있는데, 이를테면,

class strsum:

def \_\_init\_\_(self, a):

self.arg1 = a

def \_\_add\_\_(self, other): #이 함수를 불러올 명령어는 obj1 + obj2 이다.

print(self.arg1+other.arg1+”!”)

>>obj1=strsum(jack)

>>obj2=strsum(pot)

>>obj1+obj2 # jackpot!

5.2 모듈(Module)

모듈 설치 방법

모듈을 받는다. hwl 파일로. 파일 이름 중간에 cp00는 파이썬 버전 앞 두자리 # 2.7ver->cp27

C:/User/Owner에 복사(hwl 파일)

cmd 실행 (리눅스의 경우 여기부터 그냥 실행하면 됨. 없으면 다운로드)

pip install 파일이름 #실행.

pip install –upgrade 파일명 #파일 업데이트

pip list #현재 설치된 목록들

pip check 파일명 #파일 손상여부 체크.

pip download 파일명 #파일 다운로드

#pip가 안깔려 있는 경우는 설치.

<Anaconda를 사용하자>

conda list

conda search #안나오면 conda ‘모듈이름’를 구글링하면 대부분 나온다.

conda install

condat update

#원래 모듈을 설치하면 서버내 모든 파이썬 사용자에게 영향을 미친다. 호환문제를 비롯하여 여러 문제를 발생 시킬 수 있으니 개인적으로 사용해야하는데, 아나콘다가 그 개인 모듈 라이브러리를 자동으로 제공하며, 내부에 있는 라이브러리를 사용함으로써 호환성 문제에 있어 개인적으로 얼마든지 바꿔 사용하게 한다.

#conda 명령어는 아나콘다 클라우드로부터 모듈의 요구사항 체크, 버전업 등을 ‘엄청나게’ 편하게 제공한다. 갓콘다를 활용하자.

모듈이란?

변수, 클래스, 객체 등이 저장되어 있는 .py 파일을 말함. 파이썬 프로그램에서 불러올 수 있게끔 만들어진 파일. # import () 등등으로 불러와서 쓰던 것들 모두 모듈임.

모듈을 실행할려면 모듈이 있는 폴더에 들어가서 python을 실행, import 하거나 터미널에서 바로 모듈을 실행하는 경우 python module 을 입력해아하는데, 폴더로 들어가기 귀찮으므로 다음과 같은 방법들을 사용한다.

1. sys.path.append  
   파이썬 소스 코드 안에 다음 코드를 입력  
   import sys  
   sys.path.append(“모듈 경로”)
2. PYTHONPATH  
   set PYTHONPATH=모듈경로  
   #그 다음 파이썬 실행..인데 왜 인지 안됨. 모르겠음.

모듈 내에서 실행되는 것들은 불러왔을 떄 모두 실행되는데, 이때 불러왔을때 함수, 변수등 선언을 제외한 실행문들이 실행되기를 원치 않는 다면 \_\_name\_\_==”\_\_main\_\_” 조건을 사용한다.

(ex)

if \_\_name\_\_==”\_\_main\_\_”:   
 print(“this module is imported in terminal”)   
# 이 block은 dos, linux에서 python import module을 치거나 ‘%run module’ 명령어처럼 직접 실행하지 않는 한 실행되지 않음

모듈 내에 특정 함수만 불러오려면 from module import func처럼 입력하면 되며,   
from module import \* 를 사용하면 모듈 내 모든 함수를 사용하게 되나, 여러 모듈을 불러오면 같은 함수명, 변수로 인해 충돌을 일으키므로 비추.

모듈을 import 한 상태에서 해당 모듈 파일을 손대는 경우 새로 불러오는 것이 매우 번거로움. 방법을 모르면 껐다 키는 수 밖에..없지만 다음 방법을 사용하면 새로 불러와진다.

방법 :

from imp import reload

reload(mymodule)

import mymodule

>>job done!

6. 예외처리

6.1 예외(Excetion), 오류(Error)

예외:   
프로그램에서 벌어지는 오류를 포함한 예외적인 상황. 객체로서 다룰 수 있다(클래스로 선언할 수도 있고, 조건문 안에 조건으로 넣을 수도 있음). 이를 처리하는 것을 예외 처리라고 함.

명령어로 raise EX를 사용하면 EX라는 예외를 발생(raise)시킨다.

오류:   
FileNotFoundError #파일을 못찾을 때.  
ZeroDivisionError #0으로 나눴을 때.  
KeyError #인덱스가 주어진 객체의 인덱스를 벗어났을 때.  
IndexError #리스트, 튜플 등 인덱스가 있는 파일에 해당 인덱스가 없는 경우.  
AttributeError #객체에 해당되지 않는 메서드를 사용했을 때.  
TypeError #타입이 일치하지 않는 값들을 함께 처리했을 때.  
StopIteration #이더레이터가 끝났을 때.

6.2 try:

기본 형식은 다음과 같다.

try: #최소 하나의 예외가 존재해야함.  
 () #명령문  
 if ():  
 raise (EX)   
except (EX) as x: # EX는 exception class 객체. EX1,EX2, ... 처럼 묶어서 한꺼번에도 가능.  
 () # as x로 변수로 선언해 놓으면 x에 해당 EX(예외)가 입력됨.  
else:  
 () #명령문

try구문은 예외처리, 즉 except에 대해서 명령을 따로 지정한다는 점에서 의미를 갖는 구문이다. 따라서 구문 내에 except가 없으면 별다른 의미는 없다.

7. 이미지 프로세싱

이미지 프로세싱 방법은... 여러가지가 있지만 일반적으로 두가지 모듈, Image와 numpy를 이용해서 하며, 함수를 시각화하는 건 numpy & matplotlib.pylab, 사진을 가지고 RGB 분석, 처리하는 것은 PIL.Image를 이용하는 것이 정신건강에 좋다. 일단 Image를 다루어본다.

<Image>

x=Image.new(mode,size(,color)) #이미지를 생성, mode는 “RGB”,”CMYK,”L” 등이 가능.

L은 흑백모드. size와 color는 tuple.

pix=im.load() #PixelAccess 클래스의 인스턴스(이걸 고치면 im도 바뀜)를 반환. 각 픽셀에 접근 가능.

pix[0,0]=(255,255,255)

size=im.size #(x,y) : x 방향 픽셀 개수, y 방향 픽셀 개수

pix=im.load() -> PixelAccess

ImageTk.PhotoImage(image) -> PhotoImage

im.convert(mode) -> image

im.convert("L").save("graycat.bmp")

im=Image.eval(image,function) -> image

 for문을 실행할 때 시간이 상당히 오래 걸리는 것을 볼 수 있다. 일반적으로 PixelAccess 객체를 직접 이용하는 것은 상당히 느리다. 파이썬은 고급 언어이기 때문에, 단순 계산을 연속적으로 하는 이미지 프로세싱을 직접 다루기엔 다소 부적합하다. 그런 점이 PIL과 같은 라이브러리가 필요한 이유이기도 하다.

   PIL에서는 회선 마스크를 만들어주는 함수인 Kernel과 이것을 적용하는 함수인 filter를 제공한다. 다음과 같이 하면 위에 소개된 긴 코드와 똑 같은 효과로 흐린 고양이 이미지를 만들 수 있다.

import Image, ImageFilter   
cat=Image.open("cat.bmp")   
M=1.0/9.0   
boxblur=ImageFilter.Kernel((3,3),(M,M,M,M,M,M,M,M,M)) # 1/9로 이루어진 3x3회선 마스크   
cat.filter(boxblur).save("blurcat.bmp")

  부드러운 이미지 확대/축소에 사용되는 방법을 통틀어 anti-aliasing이라고 하며, 흐리게 효과를 주는 방법 이외에도 상황에 따라 많은 방법들이 사용되고 있다. PIL에서는 빠른 속도로 이미지의 크기를 변경하는 함수를 제공한다. 아래와 같이 하면 bilinear방법으로 2.5배 확대된 cat.bmp와 좋은 품질의 anti-aliasing방법으로 축소된 cat.bmp를 얻을 수 있다.

im=Image.open("d:\\cat.bmp")   
im2=Magnify(im,2.5)   
im2.save("d:\\bigcat.bmp")

|  |
| --- |
| import Image  cat=Image.open("cat.bmp")  bigsize=( int(cat.size[0]\*2.5) , int(cat.size[1]\*2.5) )  smallsize=( int(cat.size[0]\*0.5) , int(cat.size[1]\*0.5) )   bigcat=cat.resize(bigsize, Image.BILINEAR)  smallcat=cat.resize(smallsize, Image.ANTIALIAS)   bigcat.save("bigcat.bmp")  smallcat.save("smallcat.bmp") |

<Numpy & Matplotlib>

plt.imread('fname.png')

plt.imshow(z) #z는 색상을 표현할 MxN크기의 값을 갖는 배열,

#혹은 좌표별 rgba데이터를 포함한 MxNx3(or MxNx4)

plt.axis(‘off’) #축 없애기...인데 왠지 안됨.

plt.xticks(())

plt.yticks(()) #축없애기 요거 쓰자.

Scipy Lectures Note

2.4.6 Variable number of parameters # parameter : 함수 안에 들어가는 변수

Special forms of parameters:

• \*args: any number of positional arguments packed into a **tuple**

• \*\*kwargs: any number of keyword arguments packed into a **dictionary**

**In [35]: def** variable\_args(\*args, \*\*kwargs):

**....: print** 'args is', args

**....: print** 'kwargs is', kwargs

**....:**

**In [36]:** variable\_args('one', 'two', x=1, y=2, z=3)

args is ('one', 'two')

kwargs is {'y': 2, 'x': 1, 'z': 3}

**In [4]:** demo?

**In [5]:** who

**In [6]:** whos

**In [10]:** importlib.reload(demo)

**In [10]: for** dirpath, dirnames, filenames **in** os.walk(os.curdir):

**....: for** fp **in** filenames:

**....: print** os.path.abspath(fp)

**....:**

**....:**

/Users/cburns/src/scipy2009/scipy\_2009\_tutorial/source/.index.rst.swo

/Users/cburns/src/scipy2009/scipy\_2009\_tutorial/source/.view\_array.py.swp

/Users/cburns/src/scipy2009/scipy\_2009\_tutorial/source/basic\_types.rst

/Users/cburns/src/scipy2009/scipy\_2009\_tutorial/source/conf.py

/Users/cburns/src/scipy2009/scipy\_2009\_tutorial/source/control\_flow.rst

...

Numpy

Numpy?

숫자와 배열을 다루는데 특화된 파이썬 모듈. 기본으로 깔게 되는 경우가 많음. 기본적으로 파이썬은 매트랩이나 포트란처럼 숫자처리에 특화되어있는 언어가 아니지만, Numpy는 수치처리에 있어 매우 강력한 모듈로 이를 이용하면 파이썬으로도 매트랩 못지 않은 처리속도와 편리함으로 다양한 데이터 관련 작업을 할 수 있다.

#import

Anaconda를 설치하면 Numpy는 기본으로 깔리기 때문에 설치는 건너뛰고, import는 다음과 같이한다.

import numpy as np

import 문법은 다른 챕터에서 다루므로 넘어가고, 짚고 넘어가야하는 부분은 ‘as np’를 사용함으로써 numpy의 함수는 np.function을 이용해서 불러와야 한다는 점. np 대신 다른 별칭을 붙여도 좋지만 대부분 np.를 사용한다.

#Refernce doc in python

np.function? #파이썬 기본 레퍼런스 형식. 가장 많이 쓰임.

np.lookfor(‘검색어’) #검색어와 관련된 np명령어들을 모조리 불러온다.   
 #뭔 기능이 있는지조차 모르는 절망적인 상황에서 사용.  
 #ex) np.lookfor(‘dimension’)   
 #matrix dimension을 찾는 함수가 궁금할 때 검색하면 뭐가 나올꺼다.

#배열 선언법

A = np.array([[0, 1, 2, 3], [4, 5, 6, 7]]) #2행 4열 배열, np.array에서 열은 ‘리스트 안의 리스트’로  
 #표현하며, 출력할 때는 구분되서 나온다.

A = np.arange(1000) #0~999, 이때 end exclusive하다고 표현.

A.T #Transpose(전치) matrix

A = np.diag(np.array([1,2,3,4])) #[1,2,3,4]를 대각행렬로 만듦.  
 #array([[1,0,0,0], [0,2,0,0], [0,0,3,0], [0,0,0,4]])

np.eye(3) #(3,3) 단위행렬 생성.

import random #랜덤 모듈

.random() #0~1 사이의 수를 랜덤으로 선택

.choice([]) #주어진 리스트 []에 대해 하나를 선택

.randrange(a,b) #a, b-1사이의 수 중 하나를 랜덤으로 선택.

np.random.shuffle(A) #A를 랜덤으로 섞는다. (변수를 할당하는게 아니라 A가 바뀜)갖고 놀게 필요할때 쓰자.

A = np.random.rand(n) #n개 짜리 0~1사이의 수로 이루어진 array 생성

A = np.random.randn(shape) #shape=x,y 개 짜리 0~1사이의 수로 이루어진 array 생성

A = np.random.normal(mean, dev, n) #평균 mean과 표준편차 dev를 갖는 데이터 n개를 랜덤 생성.

A = np.random.uniform(min=0, max=1, size=None) #n개의 연속균등분포 랜덤배열을 생성

+) random 모듈 도태된듯. np.matlib.random. 모듈을 사용하자.

np.matlib.random.randint(min,max,size=(m,n)) #m,n 사이즈의 랜덤 배열 생성.

#View?

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

y = x[0:4]

y[0]=10

print(x)  
x = array([10,2,3,4]) #원본 자료도 수정되어 버림. y를 스칼라로 하면 안 그럼.

#np.may\_share\_memory(x, y)값이 True이면 이러는거임. 이렇게 안되게 할려면

#y=x[0:4].copy()를 하면 됨.

#이렇게 값이 할당되는 객체가 아니라 링크처럼 값만 가져와 보여주는 객체를 view라고 함.

[ 배열 다루기 ]

<Slicing>

A[ index1 : index2 ] #리스트의 그것과 비슷하다. 0부터 시작하고 끝은 포함 x

A[ start : end : term ] #여타 다른 언어에서 사용하는 것과 비슷하다.   
 #시작 index는 1, end는 포함된다.

np.s\_[start:end:term] #함수 안에서 범위를 지정해야할때,

(예를 들면 A의 일정 부분을 np.delete하고 싶다거나) 사용할 수 있는 slice method.

mask = ( A == n ) #각각 A의 원소 자리에 값이 n인가에 따라 True or False로 이루어진 배열.

#True or False를 리턴하는 조건식이라면 뭐든 넣을 수 있다.

A[ 조건식 ] #조건을 만족하는 A원소. 매우 유용!   
 #(ex) A[A % 3 ==0] == 0 는 3의 배수인 원소를 0으로 만드는 식

A[ [list] ] #list도 indexing에 활용할 수 있으며   
 #list안에 중복되는 값이 있어도 불러 올 수 있다.

A[mask] # 배열 ‘mask’에서 True인 값들만 인덱싱.

# mask = array([1, 0, 1, 0, 1])이라면

# A[mask]는 1,3,5번째 원소로 이루어진 배열이다.

A[A[:,n].argsort()] #튜플 내지는 다차원 배열을 다룰 때 n-1번째 값을 기준으로 sorting하는 방법.

A[np.argwhere(bool)] #bool(조건)을 만족하는 A를 출력.   
 np.argwhere만 하면 bool=True인 원소의 위치를 배열로 출력.  
 <nan값을 어떤 값으로 대체하고 싶을 때>  
 np.argwhere(np.isnan(A)) = np.array[nan값을 갖는 원소의 위치]  
 A[np.argwhere(np.isnan(A))]= 0 #nan값에 0을 대입

<Operation>

스칼라와 벡터간 사칙연산이 매우 자유로움. (리스트와 튜플은 엄청 깝깝하다. 편리함에서, 또 속도에서.)

기본적으로 항별 연산(Elementwise)이 이루어지며, 매트릭스간 연산에서도 그냥 수행하면 항별 연산, A@B로 명령하면 행렬곱 연산(Matrix multiplication)이 수행된다.

A == B 같은 논리연산도 가능하며, 각 항별로 같은 인덱스에 True, 다른 인덱스에 False를 넣어 배열로 출력한다.(항별 연산) 이외에도 다음과 같은 것들이 가능함. 더 있으니 매뉴얼 참고.

np.array\_equal( A , B ) #A, B가 완벽히 같으면 True, 다르면 False를 출력.

np.logical\_or( A , B ) #A, B 둘 중 하나라도 0(False)이 아니면 True, 둘다 0(False)이면

#False값을 갖는 배열 출력.

np.sin( A ) #배열 A값에 따라 sin( 원소값 )을 갖는 배열 출력.

np.triu( A , n ) #배열 A를 윗삼각행렬로 만듦.   
 #대각행렬로 부터 n만큼 떨어진 삼각행렬로, 이외에 값은 0이 된다.   
 #n은 음수값을 넣을 수도 있고, 이경우 아랫삼각행렬이 됨.   
 #아예 np.tril( A , n )을 쓰면 n만큼 떨어진 아랫삼각행렬로 만들 수도 있음.

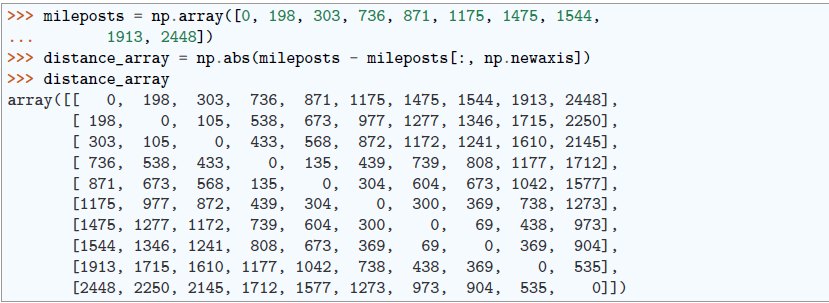
np.sum( A , axis = n ) #A의 모든 항을 더하는 것. ,axis = n 을 추가하면 n값에 따라 열/행을 더한다.  
 **#(axis 0 = Column, axis 1 = row, axis 2= z axis (if exist) ...)**

x[ : , n ].sum() #일케 해도 된다. 똑같은 짓. np의 array class로 들어가 있기 때문에 사용가능.

#단, 위 경우와 n이 의미하는 바와 결과가 다르다.   
 #조금만 생각해보면 알 수 있음. 주의할 것.

A = A – A[:, np.newaxis]#배열 A에서 세로로 배열 A를 생성해 빼는 식.

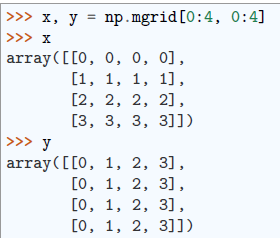
#A[:, np.newaxis]는 배열을 그리드로 만드는데 쓰인다는 점 주목.



x, y = np.ogrid[xrange, yrange] #xrange, yrange의 축을 출력,

#distance = np.sqrt(x \*\* 2 + y \*\* 2)하면 거리값의 행렬이 출력된다.

#정확한 원리는 아직 잘 모르겠음. x, y이면 튜플 아닌가? x 출력해보면 열 배열 나오던데

아니면, 그냥 이렇게도 할 수 있음.

#이건 좀 익숙하다. mgrid = meshgrid

<Sorting data>

np.sort(A, axis=n) #n은 0부터, 0번째 축을 기준으로 정렬. 0이면 x축을 기준으로, 1이면 y축을 기준으로

#a.sort로도 사용 가능.

j = np.argsort(A) #배열 A의 원소를 옵션에 따라 인덱스를 출력. 기본은 오름차순으로,

#제일 처음 나오는 인덱스가 가장 작은 원소의 인덱스이다.

#sortA = A[j]를 하면 a.sort와 같은 결과가 출력. 2차 행렬일때는 행별로 sort되는데,

#간단하게 어떻게 만들지는 잘 모르겠다.

<Polynomial>

p = np.poly1d([coeffs]) #고차항의 계수부터 배열의 수로 설정된 다항식.

p(n) #x가 n일때의 값 출력

p.roots #근을 배열로 출력

p.order #다항식의 최고차항의 차수를 출력

np.polyfit(x,y,n) (x, y) #좌표 데이터의 회귀선의 n차 다항식을 출력

p = np.poly1d(np.polyfit(x,y,n)) #위 n차 다항식을 다항식화시킨것.

#데이터와 회귀선을 그리려면 (x,y)를 플롯하고

#촘촘하게 선언한 t에 대해 (t,p(t))를 같이 플롯하면 됨.

Matplotlib

#아ㅣ아아아아아아 이거 정리 언제 다 하냐

<Plot>

**import numpy as np**

**import matplotlib.pyplot as plt**

*# Create a figure of size 8x6 inches, 80 dots per inch*

plt.figure(figsize=(8, 6), dpi=80)

*# Create a new subplot from a grid of 1x1*

plt.subplot(1, 1, 1)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)

C, S = np.cos(X), np.sin(X)

*# Plot cosine with a blue continuous line of width 1 (pixels)*

plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")

*# Plot sine with a green continuous line of width 1 (pixels)*

plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")

*# Set x limits*

plt.xlim(-4.0, 4.0)

*# Set x ticks*

plt.xticks(np.linspace(-4, 4, 9, endpoint=True))

*# Set y limits*

plt.ylim(-1.0, 1.0)

*# Set y ticks*

plt.yticks(np.linspace(-1, 1, 5, endpoint=True))

*# Save figure using 72 dots per inch*

*# plt.savefig("exercice\_2.png", dpi=72)*

*# Show result on screen*

plt.show()

*#Changing colors and line widths*

plt.figure(figsize=(10, 6), dpi=80)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")

plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")

*#Setting limits*

plt.xlim(X.min() \* 1.1, X.max() \* 1.1)

plt.ylim(C.min() \* 1.1, C.max() \* 1.1)

...

여타 프로그램과 다를바가 없다. 오버플롯은 명령어 따로 없이 여러번 쓰면됨.

[1] plot(x,y)   
[2] plot(x,z) # 혹은 plot(x,y, x,z)

#plt.subplots에 대해서.

figure, (plot1, plot2, plot3, ... ) = plt.subplots( row, colomn, figsize = (x,y), sharex=/bool/, sharey = /bool/)

plot1.set\_~ 으로 일반 plot에서 하는 대부분의 plot 설정들을 할 수 있음. (ex) plot1.set\_xlabel(‘Label’)

sharex와 sharey는 축을 공유할 경우 True, False를 할당하여 설정할 수 있음.

f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex='col', sharey='row')

이런것도 가능함! row끼리는 y를 share하고 colomn끼리는 x를 share해서 4개의 그림에서 총 네개의 축, x축, y축 두개씩을 그릴 수 있다. (원래는 4개의 그림, 각각 2개씩 총 8개 그림)

Title 설정은 subplot과 전체에 대해서 따로 할 수 있다.

fig.suptitle(‘figure title’,fontsize=xx,fontweight=’’,style=’’) #style은 이탤릭, 노멀, 오빌릭…이라는데 뭐지

ax.set\_title(‘subplot title’,위와 동일)

#데이터간 스케일 차이가 많이 나는데 한 그림에 그려야 할 때 (axis skip range)

f, (ax, ax2) = plt.subplots(2, 1, sharex=**True**)

*# plot the same data on both axes*

ax.plot(pts)

ax2.plot(pts)

*# zoom-in / limit the view to different portions of the data*

ax.set\_ylim(.78, 1.) *# outliers only*

ax2.set\_ylim(0, .22) *# most of the data*

*# hide the spines between ax and ax2*

ax.spines['bottom'].set\_visible(**False**)

ax2.spines['top'].set\_visible(**False**)

ax.xaxis.tick\_top()

ax.tick\_params(labeltop='off') *# don't put tick labels at the top*

ax2.xaxis.tick\_bottom()

#화살표나 선을 그려야 할 때.

plt.annotate(“text”, xy=(x0,y0), xytext=(x1,y1), arrowprops=dict(arrowstyle=”->”, connectionstyle=”arc3”))

x0,y0는 도착지점, x1,y1은 시작지점(텍스트가 위치할 지점이기도 함) ->는 시작->끝, <-로 하면 반대도 가능. Arc3는 곡률인 모양인데 그냥 이거 쓰셈.

# What is LaTeX, and how to?  
LaTeX(ˈleɪtɛk 레이텍 또는 ˈlɑːtɛk 라텍)은 문서 조판에 사용되는 프로그램이다. 도널드 커누스가 만든 TeX을 쉽게 사용하기 위하여 1984년에 레슬리 램포트가 만든 매크로이다. From Wiki…

from matplotlib import rc

rc(‘font’, \*\*{‘family’:’sans-serif’,’sans-serif’:[‘Helvetica’]})

rc(‘text’, usetex=True)

…잠깐.

<https://matplotlib.org/users/mathtext.html>

엄… rc를 왜써야 하는지 모르겠다. 이거 참조할 것.

<요약>

‘text’ 앞에 r을 붙일 것, 특수문자 구문 시작할 때, 끝날 때 $를 붙일 것, 특수문자 앞에 \를 붙일 것.

Ex)

r‘Nu is $\nu$.’

#Basemap  
지도 그리는 툴킷. from mpl\_toolkits.basemap import Basemap으로 임포트.

m = basemap()으로 기본적으로 사용함. 이 설정 다 해놓고 plt.show()하면 결과값 출력.

cb =

ok let’s do this

Spyder : Scientific Python Development Environment

Jupyter notebook과 비슷한, 파이썬 개발 환경 제공 프로그램. MIT 제공.

Conda install spyder로 설치. (-t를 붙였던가? 아무튼 그럼)

#F5가 Run, 전체적으로 IDL과 비슷한 환경이다.

#ctr+shift+I 는 console창으로 이동, ctr+shift+E는 editor창으로 이동.

#F12가 break point!

#debug 모드에서는 한 함수만 실행하거나 break point까지만 실행하는 등 코드에 오류가 있을 때 매우 유용한 툴들이 많다. Run을 하려면 debug 모드를 종료하고 해야함.

#

## 단편 지식

np.allclose( A, B, rtol=1e-05, atol=1e-08, equal\_nan=False)

#두 배열 A, B가 원소별로 비교했을 때 허용치(tolerance) 안에서 같은가에 대해 논리값(bool)으로 돌려준다. 허용치에서 벗어날 정도로 다르면 False, 허용치는 설정이 가능하다.

<http://regexr.com/> 정규표현식 연습페이지

\ #ㅑ로 넘기기

<연산자 정리> #man operator참조

a%b # a를 b로 나눈 나머지

a|=b # a or b

a//b #a를 b로 눈 몫

empty((x)) #x개의 임의의 작은 행렬을 만듦. 괄호 빼도 똑같음. 뭐지?

import timeit

timeit.timeit(function) #function 수행하는데 걸리는 시간 확인.

import time

time.sleep(n) #n초 만큼 멈췄다가 계속함.

%reset #ipython 에서 clear, 안될 때도 있다. pycharm에선 모르겠다. 복잡하다.

[이스케이프 코드란?] #문자열(str type)에서 특수문자를 사용해야 할 때 사용하는 코드.

코드 설명

\n 개행 (줄바꿈)

\t 수평 탭

\\ 문자 "\"

\' 단일 인용부호(')

\" 이중 인용부호(")

텍스트 모드 vs 바이너리 모드 : 전자는 글자로 읽고 후자는 이진법으로 읽는다. 파일 열때 후자로 열려면 wb,rb 등을 써야함.

os 모듈

시스템마다 다를 수 있는 문법들을 호환시켜주는 모듈. 파일경로를 지정할때, 혹은 디렉토리 목록을 검색하고 리스트를 출력할 때 매우 유용함. 특히 윈도우와 리눅스는 디렉토리 구분 문법이 다른데, os.path를 이용하면 둘다 호환이 되기 때문에 리눅스에서 짠 코드가 윈도우에서 바로 사용할 수 있게 된다.

os.path.abspath(‘file’) #file의 ‘경로’ 문자열을 출력.

os.path.split(‘경로’) #디렉토리와 파일명으로 이루어진 튜플을 출력.

os.path.dirname(‘경로’) #경로까지만 뽑아서 출력

os.path.join(‘경로’,’파일명’) #경로와 파일명을 묶어서 파일의 위치 출력.

os.getcwd() #현재 디렉토리의 문자열을 출력.

os.walk(‘\*/\*/\*’)

os.path.isdir(‘\*,\*’) #논리 변수 출력

#디렉토리 내 모든 폴더 싹다 뒤지는 코드

In [50]: def Type2():

...: for root, dirs, files in os.walk(os.getcwd()):

...: for file in files:

...: if file.endswith(".txt"): #출력할 파일 조건. 다른 검색 조건은 str.[tap] 참조

...: print(os.path.join(root,file)) #이거 말고 os.path.abspath(file))해봤는데 안됨...

더 자세한 건 도움말 참조.