클로저 쓰는 이유 중 하나는 은닉.

라이브러리 같은게 모듈 자바는 임포트?

코드들 묶어놓은거

확장자 .pyd가 모듈

함수를 포함하고 있는 형태의 모듈

별도로 사용도 가능하고 모아서 사용도 가능함

라이브러리라고 보면됨

표준모듈이 모든걸 다 제공하지는 않음

없으면 서드파티 찾아서 하면 됨. 찾아서 설치

cmd에서 py가 있는 곳으로가서

maroroWorkspace> python 명령어로 파이썬 실행

>import mymath

>dir(mymath)

import가 모듈추가하는 명령어

파일명을 모듈명으로 바로 쓸 수 있고 .py는 생략

dir로 정의된 이름들 보기

print(mymath.mypi) 쓰려는 변수, 함수 앞에 모듈명을 써줘야함

import os os는 표준 모듈 os.path.join os 모듈안에 path모듈안에 join

자격변수 무자격변수???????

>>> import sys

>>> sys.path

['', 'C:\\Python\\Python36\\Lib\\idlelib', 'C:\\Python\\Python36\\python36.zip', 'C:\\Python\\Python36\\DLLs', 'C:\\Python\\Python36\\lib', 'C:\\Python\\Python36', 'C:\\Python\\Python36\\lib\\site-packages']

여기에 workspace 주소가 없으면 실행이 안됨 import할 때 검색 대상공간이됨

그래서 추가해줘야함 sys.path.append(‘c:\\’)

절대 import와 상대 import가 있다

.으로 시작하지 않는 것은 모두 절대 import

from mymath import add

이렇게 하면 from다음에 모듈명이나 패키지명이 옴 import 다음에는 해당 모듈이나 패키지 안에 있는 변수나 함수 명

import mymath as my

my.add(5,3) ㅇ런식으로 쓸 수 있음

문자열로 받은 모듈명 import하라면

re=\_\_import\_\_(modulename) 이렇게 하면됨

\_\_import\_\_(modulename)

한번 import된 모듈은 다른 모듈에서 가져오기 해도 이미 import 됨 모듈과 공유함

다시 강제로 import 하고 싶으면 imp모듈의 reload함수 사용

import imp

imp.reload(math)

이런식. 모듈 업데이트 할 떄 주로 사용. 라이브러리 개발할 때

if \_\_name\_\_=='\_\_main\_\_':

print(add(5,3))

print('mypi=',mypi)

print(area(5))

print(\_\_name\_\_) #이러면 직접 실행했을 때 \_\_main\_\_나오는거 확인

모듈 파일에서 이렇게 해주면

main일 때만 직접 실행했을 때만 print되도록 해준거

C는 exe만드는데

자바는 바이트코드 만들어서 가상머신에 넘겨서 실행 .exe파일 안만듬

python도 비슷함 exe파일 안만듬

cmd에서 직접 스크립트파일 실행하는게 실제 단독 실행.

import 해서 하면 단독 실행아님

스크립트에서 F5는 그냥 테스트환경. 개발 단계

단독으로 실행 하는 것만 main

패키지는 모듈을 모아 놓은 단위

성격에 따라서 패키지 관리??????

패키지 만들 때 \_\_init\_\_.py 있어야 나중에 패키지로 찾아서 관리 할 수 있음

패키지에는 디렉토리 경로가 있을 수 있다

디렉토리 하나하나가 네임스페이스

절대 경로는 .으로 시작하지 않음. 상대경로만 . 씀

.은 현재

distutils을 이용하여 배포파일 만듬

배포파일은 setup.py파일 있어야함

py2exe.org

여기서 다운 받아서 따라하면 실행파일 만들 수 있음

프레임워크는 라이브러리 집합, 구조라고 보면됨 framework를 활용하면 개발이 더 쉬움

클래스가 있어야 객체생성 가능함

클래스느 속성이랑 행위로 구성되어있음

속성(attribute)는 상태 값.

행위(behavior)은 상태에 대한 변화 동작

리눅스는 C로 구현했는데 객체지향 기법으로 구형함.

C자체는 객체지향언어는 아닌데 구현은 할 수 있음. 코딩량 늘어남

클래스도 하나의 이름공간

클래스를 정의하는 것은 하나의 새로운 자료형을 만드는 것

클래스는 하나의 템플릿 틀

틀을가지고 인스턴스 여러 개 만들 수 있음

클래스 이름 공간과 인스턴스 이름 공간은 다름

함수호출 식으로 인스턴스 생성. java는 new 썼는데 파이썬은 그냥 x=S1()이런식

파이썬에서는 클래스, 인스턴스가 독립된 이름 공간

추상화 : class 도출을 위한 방법

캡슐화 : 묶어준다 (관련있는 것을)

다형성 : 상황에 따른 적절한 동작

상속 :재사용성

부모 역할을 하는 클래스를 base, parent, super class라고 함

상속 받는 클래스를 Derived, child, sub class라고 함

class B(A): 이렇게하면 A를 상속 받은거

재사용성이 높다?

class다음에 쓰는걸 클래스 변수

생성자(constructor) : Instance생성시 자동 호출 함수

소멸자(destructor) : Instance 소멸시 자동 호출 함수

생성자 이름을 \_\_init\_\_

def \_\_init\_\_(self,[,인수1, 인수2]):

\_\_def\_\_는 소멸자

def \_\_del\_\_(self);

소멸자는 알아서 가비지 콜렉터가 해주기 때문에

자바에서 this랑 비슷한게 self

self: class 함수의 첫 인수 이름

self있으면 클래스소속함수

첫번째는 무조건 self 소속 함수면

두번째 부터터는 마음 대로

클래스 함수는 갯주 제한 없이

base는 기초클래스, 슈퍼클래스

object:python 최상위 class

s1=simple()

에서

s1은 인스턴스 변수명 (인스턴스의 레퍼런스 저장)

whitebox 형식 (외부에서 안이 보이는거)

blackbox(ㅂ외부에서 안이 안보임) –private

파이썬은 그런게 없음

whitebox로 관리. 외부에서 다 보임

클래스 이름공간과 인스턴스 이름 공간은 별도

동적으로 외부에서 멤버를 생성할 수 있다.

s1=simple()

s1.stack=[]이런식으로

>>> class MyClass:

def set(self,v):

self.value=v

def get(self):

return self.value

>>> c=MyClass()

>>> MyClass.set

<function MyClass.set at 0x0000000002FA6F28>

>>> MyClass.set(c,'egg')

>>> MyClass.get(c)

'egg'

첫번쨰 인수로 self가 전달되고 있으면 메서드, 아니면 일반 함수

self로전달되는 인수를 인스턴스 변수???????

클래스이름이 메서드 앞에 나오면 셀프 값을 넣어줘야 함

클래스명으로 클래스함수(매서드) 호출하려면 첫번쨰 인수에 반드시 셀프 값 넣어줘야함. 그래서 인스턴스로

클래스명으로 호출하는걸 언바운드 메서드

>>> print(c.get())

egg

인스턴스명으로 호출한거는 셀프 안넣어줘도 됨. c자체가 self니깐. 바운드 메서드 호출

클래스명은 틀임 실체가 아니고.

self는 결국 인스턴스 정보. 그래서 무조건 인스턴스

>>> MyClass.set(c,'egg')

>>> MyClass.set(d,'egg')

>>> d.get()

'egg'

>>> c.get()

'egg'

>>> c.set('spam')

>>> c.get()

'spam'

>>> d.get()

'egg'

인스턴스 생성할 떄 자동을 hal리 준비해야하는걸 생성자에서 함.

>>> c3=MyClass2()

>>> print(c3.get())

Traceback (most recent call last):

File "<pyshell#136>", line 1, in <module>

print(c3.get())

File "<pyshell#128>", line 5, in get

return self.value

AttributeError: 'MyClass2' object has no attribute 'value'

set()으로 value값을 안만들어줘서 에러 발생.

그래서 인스턴스 객체가 생성될 때 초기화를 해줌. 자동으로 호출

생성자나 소멸잘르 임의로 호출 하는건 안됨

>>> class MyClass3:

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

self.value=0

def set(self,v):

self.value=v

def get(self):

return self.value

>>> c3=MyClass3()

>>> c3.get()

0

>>> class Life:

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

self.birth=ctime();print('birthday',self.birth)

def \_\_del\_\_(self):

print('Deathday',ctime())

>>> life=Life()

birthday Fri Sep 15 15:41:13 2017

>>> del life

Deathday Fri Sep 15 15:41:30 2017

>>> life=Life()

birthday Fri Sep 15 15:44:14 2017

가비지컬렉터가 있기 때문에 소멸자는 잘 안씀

소멸 할 때 마무리 할 작업들 지정

메모리나 자우너 관리가 자동으로 이루어지기 때문에