기초 클래스 생성자의 초기값을 파생클래스의 생성자가 받아줘야함

실행 시점에 어떤 함수가 동작할지 결정

파이썬은 RTTI ( 실행시간 type검사) 안함- 어떤 함수가 호출 될지 실행시간에 검사할 필요 x

추상화 레벨이 높다 -> 공통 정보만 가지고 있다

자바는 다중상속이 안되지만 인터페이스로 유사하게 표현 가능

다중 상속은 가능한 안하는게 좋음.

파이썬은 복잡한 구조가 아니라 다중 상속에 있어서도

다중 상속 할 때 괄호 안에 슈퍼클래스 명시한 것중 왼쪽것부터 init해주기(생성자 호출)

검색 순서도 왼쪽거부터

>>> class A:

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

pass

>>> class B:

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

pass

>>> class AA(A): pass

>>> class BB(B): pass

>>> class C(AA,BB): pass

>>> c=C()

>>> C.\_\_mro\_\_ #C.mro()

(<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.AA'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.BB'>, <class '\_\_main\_\_.B'>, <class 'object'>)

다중상속 가능한 하지 말고 상위꺼 중복되지 않게 예를들면 AA랑 BB랑 둘다 함수 f()가 있으면 뭐를 가져다 써야할지 모르니깐

>>> import inspect

>>> inspect.getmro(C)

(<class '\_\_main\_\_.C'>, <class '\_\_main\_\_.B'>, <class 'object'>)

상속 구조 볼 수 있음

다중 상속은 누구를 검색해서 찾느냐가 중요.

검색 순서가 중요

>>> class A: pass

>>> class AA(A): pass

>>> class B: pass

>>> class BB(B): pass

>>> class C(AA,BB):pass

>>> import inspect

>>> import pprint

>>> pp=pprint.PrettyPrinter(indent=4)

>>> pp.pprint(inspect.getclasstree([C]))

[ (<class '\_\_main\_\_.AA'>, (<class '\_\_main\_\_.A'>,)),

[(<class '\_\_main\_\_.C'>, (<class '\_\_main\_\_.AA'>, <class '\_\_main\_\_.BB'>))],

(<class '\_\_main\_\_.BB'>, (<class '\_\_main\_\_.B'>,)),

[(<class '\_\_main\_\_.C'>, (<class '\_\_main\_\_.AA'>, <class '\_\_main\_\_.BB'>))]]

pprint는 옵션을 줄 수 있음. 속성 값을 추가 가능

class

Object -> 속성+행위

Abstract

encapsulation 연관 있는 내용을 모아서 구성

information hiding

inheritance

polymorphism(다형성) 상속도 다형성에 한형태

-function overloading(중복 정의)

-function overriding(재정의)

-template(틀, run time 시 type결정)

다형성- 여러형태를 가지는거. 산술연산+랑 문자열 + 리스트 +의 동작이 다른거 적용하는 타입에 따라 결과가 달라진다는거가 다형성

파이썬은 화이트 박스. 외부에서 자유롭게 접근 가능

파이썬은 정보은닉은 적용할 수 없음

위임(delagation)은 상속이랑은 다름. 어떤 객체가 자신이 처리할 수 없는 메시지를 바등면 해당 메시지를 처리할 수 있는 다를 객체에 전달 하는 것

>>> class Delegation:

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

self.stack=data

def \_\_getattr\_\_(self,name):

print('Delegation {} '.format(name),end=' ')

return getattr(self.stack,name)

>>> a=Delegation([1,2,3,1,5])

>>> print(a.pop())

Delegation pop 5

>>> print(a.count(1))

Delegation count 2

pop이나 이런거 없는데 위임해서 처리된거

syntax error :문법 error 컴파일 단계

link error: library함수 연결 error link단계

warning: 잠재적인 error 가능성이 있다. compile 단계에서 option 조절

이 세개는 cimpile단계에서 발생

run time error: 실행시 발생하는 모든 에러가 런타임 에러. 값이 틀려도. 명시적인 에러 메시지뿐만 아니라 결과에 문제가 생긴 것도 발생시 program 강제 종료. 정상종료 될 떄 처리되어야 했던게 아무 것도 안이뤄짐

다음 번 프로그램 실행 될 때 문제가 될 수 있음

예외처리가 되는 런타임 에러가 있음. 예외처리라는건 구조화를 해야함

구조적인 예외처리가 안되는 언어 - C언어. if문을 써서 처리해야하는데 지저분해짐

python documentation(f1키) – python standard library- built in exception –exception hierarchy

try-except문으로 예외를 잡아낼 수 있음. 비정상적인 상황 발생 시 강제 종료 돼도 상관 없으면 처리 안해도 되는데 반드시 잡아줘야하는게 있음. 매뉴얼에 있음,. 무조건 처리해줘야함

>>> try:

print(10.0/0)

except IOError:

print('IO Error')

Traceback (most recent call last):

File "<pyshell#103>", line 2, in <module>

print(10.0/0)

ZeroDivisionError: float division by zero

traceback저런게(빨간 글씨 에러) 강제종료 되는거. 예외처리 안된거

>>> try:

print(10.0/0)

except Exception:

print('Exception!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!')

Exception!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

이래도 예외처리 되는게 exception이 zerodivision의 상위 클래스?여서

모든 예외처리하려면 BaseException 혹은 except 뒤에 아무것도 안써주면 됨

>>> try:

spam()

except NameError as x:

print(x)

name 'spam' is not defined #이미 지정되어 있는 출력문 형태가 나옴

>>> def this\_fails():

x=1/0

>>> try:

this\_fails()

except ZeroDivisionError as err:

print('Runtime error:',err)

Runtime error: division by zero

except ( TypeError, IOError):# 두예외는같은루틴을공유한다.

이렇게 묶어도 됨. 그러면 예외처리 실행 구문이 같음

raise는 우리가 직접 에러를 발생시킬 필요가 있을 때

그냥 raise만 쓰면 바로 직전에 발생한 예외

raise 다음에쓰면 그거.

assert: 조건을 통과하지 못하면 강제 종료

TDD(Test driven development- 개발 방식 중 하나) 프로그램 개발 방식을 test로 하는거

문제 ->test case(test code: 통과전)🡪통과 후

test case를 얼마나 잘만드냐가 관건 framework Junit????????

파일 종류

data 형식

text 파일

binary 파일 보안????????/

record 형식 - record: 파일 입/출력 단위

고정길이 record 낭비가 있을 수 있음. 다 안차있을 수 있어서. 길이변화 심할 때는 안좋음

가변길이 record

data 저장/검색 방식

직접 파일

순차파일

색인 순차 파일

순차 파일은 저장 순서대로 기록? 처리가 쉬움 but 검색이 어려움

색인 순차 파일은 순차로 저장하는데 색인 파일 따로 만들어서 색인 파일의 레코드와 호환된 레코드의 위치에 접근해서 읽어오거나 기록할 수 있는 장점

직접파일은 바로 찾아가면 되니깐 쉬운데???? 구현이 어려움

r+읽기에 갱신도 가능 업데이트용???????????

모든 파일은 BOF(begin of file)에서 싲가해서 EOF에서 끝남. 바이트 단위로 저장

file pointer는 read모드에서는 맨 처음에 append는 맨 마지막에 위치함. 그리고 리드 할 때 마다 record만큼 옮겨감

recordpointer???????

기존 파일 있는데 w모드하면 덮어쓰기 돼서 기존 파일 사라짐

filemode 생략하면 기본적으로 text파일

>>> f=open('C:\maroroWorkspace\\file.txt','r')

>>> print('파일 현재 위치:',f.tell())

파일 현재 위치: 0

>>> print(f.read())

1 번쨰 줄입니다.

2 번쨰 줄입니다.

3 번쨰 줄입니다.

4 번쨰 줄입니다.

5 번쨰 줄입니다.

6 번쨰 줄입니다.

7 번쨰 줄입니다.

8 번쨰 줄입니다.

9 번쨰 줄입니다.

10 번쨰 줄입니다.

11 번재 줄입니다.

12 번재 줄입니다.

13 번재 줄입니다.

14 번재 줄입니다.

15 번재 줄입니다.

16 번재 줄입니다.

17 번재 줄입니다.

18 번재 줄입니다.

19 번재 줄입니다.

>>> print('파일 현재 위치:',f.tell())

파일 현재 위치: 361 #파일 크기

>>> print()

>>>

f.seek(0)#처음으로

0

>>> print('파일 현재 위치:',f.tell())

파일 현재 위치: 0

>>> print(f.readline(),end='')

1 번쨰 줄입니다.

>>> print('파일 현재 위치:',f.tell())

파일 현재 위치: 19

>>> f.close()

파일 크기 보면 제어판에서 보면 1kb로 나와있는데 내꺼 크기는 361바이트 1kb는 최소 단위 클릭해서 아래보면 정확히 크기가 나와있긴 함. 운영효율을 위해서 잡아둠

읽지 않고 바로 마지막 값 읽으려면 ‘a’모드로 하면 append라 마지막으로

with open('C:\maroroWorkspace\\foo.txt','w') as f:

f.write("LIfe is too short, you need python")

이렇게 with문 쓰면 close 자동으로 됨

exception처리 안해도 열리지를 않아서 괜찮음????

모드 생략하면 read

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

with open('C:\maroroWorkspace\\test.bin','wb') as f:

f.write(bytes(data))

열어보면 이상한 값인데 아스키형식에 맞지 않은걸로 나옴

메모리에는 맞게 들어가있음. text에디터로 읽어봐야 의미 없음

바이너라 파일은 바이트 타입으로 저장. 읽어 올 때도 바이트 타입으로 읽어옴

png파일을 뷰어로 안띄우고 그냥 읽으면 png파일 크기만큼 바이트 값 나옴

dump는 파일에 기록할 떄?

load는 읽어 올 때