

```

print("Horizon-----")
print("COPY")

# Scala 데이터 참조
a = 1
print(id(a))
print()

a = 2
print(id(a))
print()

# a에 저장된 데이터가 다르기 때문에 다른 id 값을 가진다

# 저장해 둔 1 을 가리키는 것이기 때문에 a가 1 일때 id 값과 동일하다
b = 1
print(id(b))
print()

# 지금 이 code 에서
a = 10
# b 는 10 이되고
b = a

# a 는 20 을 가리키기 때문에
a = 20

c = 10
print(id(a))
print(id(b))
print(id(c))

# 가리키는 위치가 다르다
print(id(a) == id(b))

# c또한 가리키는 곳이 10 이고 , b 도 10을 가리키기 때문에 id 값이 일치한다.
print(id(c) == id(b))

```

```

Horizon-----
COPY
140541262227760

140541262227792

140541262227760

140541262228368
140541262228048
140541262228048
False
True

```

## → vector data

: 참조를 복사하는데

참조를 이용해서 세부 데이터를 변경하면 동일한 참조를 가지고 있어도  
데이터가 영향을 받는다.

```
print("Horizon-----")
print("Python 의 Vector ")
print()

ar = [100, 200, 300]
br = ar
print()

# 세부 데이터를 변경하였으므로 br 에 영향을 준다
print(ar)
print(br)
print()
ar[0] = 300
print(ar)
print(br)
print()

# ar 의 참조 자체가 변경된 것이므로 ar 과 br 은 아무런 관계가 없다
print(ar)
print(br)
print()
ar = [300, 200, 100]
ar[0] = 700
print(ar)
print(br)
```

```
Horizon-----
Python 의 Vector

[100, 200, 300]
[100, 200, 300]

[300, 200, 300]
[300, 200, 300]

[300, 200, 300]
[300, 200, 300]

[700, 200, 100]
[300, 200, 300]
```

→ copy 모듈의 copy method 는 얇은 복제를 지원하는 method

얇은 복제( 재귀적으로 복제를 하지 않음)를 지원하는 method

deepcopy( 재귀적으로 복제)

: 깊은 복제를 지원하는 method

```

print("Horizon-----")
print("COPY Module")
import copy

ar = [100, 200, 300]

# 얽은 복제를 이용 - 새로운 곳에 복사를 해서 그 곳의 id를 리턴
br = copy.copy(ar)
print(ar)
print(br)

print('middle line----')
# 새로 복제를 했기 때문에 ar 을 변형시켜도 br 에는 영향이 없음
ar[0] = 800
print(ar)
print(br)

# 문제가 되는 경우
print('middle line 2 ----')
ar = [[1, 2, 3], [4, 5, 6]]

# list안에 다른 list가 존재하는 경우 copy로 복제를 해도
# 내부 데이터를 변경하면 복제된 데이터에 영향을 준다
br = copy.copy(ar)
print(ar)
print(br)
ar[0][0] = 8920
print("result -----")
print(ar)
print(br)

#list 안에 다른 list가 존재하는 경우 deepcopy 로 복제하면
# 재귀적으로 복제를 하므로 위와 같은 현상이 없어진다
br = copy.deepcopy(ar)
print(ar)
print(br)
ar[0][0] = 989999999
print("result -----")
print(ar)
print(br)

```

```

Horizon-----
COPY Module
[100, 200, 300]
[100, 200, 300]
middle line----
[800, 200, 300]
[100, 200, 300]
middle line 2 ----
[[1, 2, 3], [4, 5, 6]]
[[1, 2, 3], [4, 5, 6]]
result -----
[[8920, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]
result -----
[[989999999, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]

```

# Have To Study Thing

# 〈 Python의 memory 정리 〉

python의 인스턴스는 **reference count** 기반의 **memory 정리 기법**을 사용

garbage collection 이 **reference count**를 확인해서  
**Instance의 memory**를 정리

처음 Instance 가 생성될 때, **reference count** 는 1이 되고  
이 Instance 를 다른 변수가 참조하면 **reference count** 는 1 이 증가

Instance를 가리키는 변수가 없어지거나  
다른 인스턴스를 가리키면 **reference count** 는 1 이 감소

**reference count**가 0 이 되면 Instance memory 영역은 자동 소멸

```
print("Horizon-----")
print("Memory 정리")
```

```
class Temp:
    # Instance 가 memory에서 소멸될 때 호출되는 함수 --> 소멸자
    def __del__(self):
        print("Temp 클래스의 인스턴스가 파괴 된다.")
```

```
# 인스턴스가 생성 reference count 가 1
obj = Temp()
```

```
# 인스턴스를 다른 변수가 가리키도록 하는 것
# obj1 = obj
# reference count 가 1 증가한다
```

```
obj = 1
# 이전에 만들어진 인스턴스 대신에 새로 만들어진 인스턴스를 가리킨다
# 이전에 만들어진 인스턴스의 reference count 는 1 감소한다 .
```

## → weakref 모듈의 ref 함수를 이용해서 참조를 복사하면

```
print("middle-----")
#참조 횟수가 1
obj = Temp()
```

```
# 참조를 복사했으므로 참조횟수 2
obj1 = obj
```

```
# 다른 인스턴스를 참조했으므로 참조횟수가 1 줄어들어 1이 된다.
obj = Temp()
```

# < Queue 모듈 >

## Queue : FIFO( First In First Out )

python에서는 queue 모듈에서 Queue, Priority Queue, Stack 을 제공

queue 모듈의 클래스들은

멀티 thread 환경에서 안전하도록 설계되어 있기 때문에  
동시에 데이터를 저장하고 꺼내도 정상적으로 작동하는 것을 보장

```
print("----- Horizon")
print("Queue 모듈 ")
import queue

li = ['a', 'b', 'c', 'd']

# Que를 생성해서 데이터를 추가
alphabet = queue.Queue()

for x in li:
    alphabet.put(x)

print(alphabet)

# que 에서 꺼낼때는 get() 을 사용
print(alphabet.get(0))
print(alphabet.get(0))
print(alphabet.get(0))
print(alphabet.get(0))

print('----- PriorityQueue = 우선 순위 큐 ')
li1 = ['다', '하', '사', '가']

hangle = queue.PriorityQueue()
for x in li1:
    hangle.put(x)
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))

print('-----LifoQueue = 스택 ')
li2 = ['다', '하', '사', '가']

hangle = queue.LifoQueue()
for x in li2:
    hangle.put(x)
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))
```

```
----- Horizon
Queue 모듈
<queue.Queue object at 0x7ff2b007f640>
a
b
c
d
----- PriorityQueue = 우선 순위 큐
가
다
사
하
-----LifoQueue = 스택
가
사
하
다
```

FRI

Back End 나 Front End 개발에서는 Thread 개발이 중요

Operator(운영), Data Mining을 하고자 할 때는 Thread 의 개념이 중요

## \*\* Thread

—> Mutual Exclusion(상호배제) -Mutex

- . 멀티 Thread 사용 시 발생하는 문제
- . 하나의 Thread 가 **사용중인 공유 자원**을 다른 Thread가 수정하는 문제

이와 유사한 문제로  
Server 에서 DataBase를 사용할 때  
Transaction에 대한 옵션을 설정하는 부분

Balance - 10000

### A Thread

Balance 조회  
Balance 수정  
Balance 조회

### B Thread

Balance 조회 —> 조회는 동시해 해도 괜찮음  
Balance 수정 —> 수정하는 작업은 병렬 처리를 하지 않음  
Balance 조회

\* 병렬과 구분을 해야함

Multi Thread : 하나의 작업중 잉여시간이 있을 때 , 다른작업을 수행

연산작업 —> CPU  
통신 —> Memory

**수정하는 작업은 병렬로 처리하면 안됨**  
—> 사공이 많으면 배가 산으로 가는 느낌 그래서 직렬로 처리함

### A Thread

Balance 조회 - 1.  
Balance 수정 - 3.  
Balance 조회

### B Thread

Balance 조회- 2.  
Balance 수정- 4.  
Balance 조회

1. 조회 10000
2. Delay로 인해서 B Thread 가 발생
3. A Thread 에서 - 4000 ,
4. 3 번 실행 시 , Delay 가 발생해서 B Thread 가 발생 , -8000 을 함

: 이때 3이 완료 되지 않아 Balance 가 - 10000이 된다 .

그렇기 때문에 조회를 제외한 삽입, 삭제 , 수정 시 @Transactional, 2개의 테이블 조회

## Thread 와 Transaction 의 중요성

### - 해결방안

### Synchronized

### Lock

```
# Threading
import threading
import time

print("----- 공유자원 수정 문제 발생 ")

# Sleep 을 기재하는 이유
# 수업중에 하는 것들은 Delay 가 발생하기 어렵기 때문에 추가하는 것
# 실제로 Thread 사용시 Sleep 을 사용하지 않을 수 있다.

# 공유 자원을 생성
g_count = 0

print("Thread Class 상속 ---- Thread Class 생성 ")
# 파이토치에서 상속받아서 많이 사용
# 자유도가 높다는 장점이 있다 .
class ThreadEx(threading.Thread):
    def run(self):
        # 파이썬에서 함수 외부의 변수를 사용
        global g_count

        for i in range(0, 10, 1):
            print('id={0} 증가하기 전 ---> {1}'.format(self.getName(), g_count))
            g_count = g_count + 1
            # Thread 전환을 위해서 잠시 대기
            time.sleep(0.5)
            print('id={0} 증가하기 전 ---> {1}'.format(self.getName(), g_count))
            time.sleep(0.5)

# Thread를 생성해서 실행

for i in range(2):
    th = ThreadEx()
    th.start()
```

```
----- 공유자원 수정 문제 발생
Thread Class 상속 ---- Thread Class 생성
id=Thread-1 증가하기 전 ---> 0
id=Thread-2 증가하기 전 ---> 1
id=Thread-1 증가하기 전 ---> 2
id=Thread-2 증가하기 전 ---> 2
id=Thread-1 증가하기 전 ---> 2id=Thread-2 증가하기 전 ---> 2

id=Thread-1 증가하기 전 ---> 4id=Thread-2 증가하기 전 ---> 4

id=Thread-1 증가하기 전 ---> 4
id=Thread-2 증가하기 전 ---> 5
id=Thread-1 증가하기 전 ---> 6
id=Thread-2 증가하기 전 ---> 6
id=Thread-1 증가하기 전 ---> 6
id=Thread-2 증가하기 전 ---> 7
id=Thread-1 증가하기 전 ---> 8
id=Thread-2 증가하기 전 ---> 8
id=Thread-1 증가하기 전 ---> 8
id=Thread-2 증가하기 전 ---> 9
id=Thread-2 증가하기 전 ---> 10id=Thread-1 증가하기 전 ---> 10
```



## → Python에서는 Lock 클래스와 RLock 클래스를 이용해서 해결

### → Lock 클래스

acquire()

: 이 method 가 호출되면 release() 를 호출할 때 까지 Thread 에게 제어권을 넘기지 않음

### → RLock()

: acquire()를 여러번 호출할 수 있는 클래스

```
# Threading
import threading
import time

print("----- 공유자원 수정 문제 발생 ")

# Sleep 을 기재하는 이유
# 수업중에 하는 것들은 Delay 가 발생하기 어렵기 때문에 추가하는 것
# 실제로 Thread 사용시 Sleep 을 사용하지 않을 수 있다.

# 공유 자원을 생성
g_count = 0

# 상호배제를 위한 인스턴스 생성
lock = threading.Lock()

print("Thread Class 상속 ---- Thread Class 생성 ")
# 파이토치에서 상속받아서 많이 사용
# 자유도가 높다는 장점이 있다 .
class ThreadEx(threading.Thread):
    def run(self):
        # 파이썬에서 함수 외부의 변수를 사용
        global g_count
        global lock

        for i in range(0, 10, 1):
            # 다른 Thread 가 작업할 수 없도록 한다
            lock.acquire()
            print('id={0} 증가하기 전 ---> {1}'.format(self.getName(), g_count))
            g_count = g_count + 1
            # Thread 전환을 위해서 잠시 대기
            time.sleep(0.5)
            print('id={0} 증가하기 후 ---> {1}'.format(self.getName(), g_count))
            # 다른 Thread 가 작업할 수 있도록 한다
            lock.release()
            time.sleep(0.5)

# Thread를 생성해서 실행
for i in range(2):
    th = ThreadEx()
    th.start()
```

```
id=Thread-2 증가하기 후 ---> 10
id=Thread-1 증가하기 전 ---> 10
id=Thread-1 증가하기 후 ---> 11
id=Thread-2 증가하기 전 ---> 11
id=Thread-2 증가하기 후 ---> 12
id=Thread-1 증가하기 전 ---> 12
id=Thread-1 증가하기 후 ---> 13
id=Thread-2 증가하기 전 ---> 13
id=Thread-2 증가하기 후 ---> 14
id=Thread-1 증가하기 전 ---> 14
id=Thread-1 증가하기 후 ---> 15
id=Thread-2 증가하기 전 ---> 15
id=Thread-2 증가하기 후 ---> 16
id=Thread-1 증가하기 전 ---> 16
id=Thread-1 증가하기 후 ---> 17
id=Thread-2 증가하기 전 ---> 17
id=Thread-2 증가하기 후 ---> 18
id=Thread-1 증가하기 전 ---> 18
id=Thread-1 증가하기 후 ---> 19
id=Thread-2 증가하기 전 ---> 19
id=Thread-2 증가하기 후 ---> 20
```

## < 생산자 와 소비자 문제 >

1. 생산자 thread : 소비자 Thread 가 사용할 자원을 만들어주는 Thread

2. 소비자 thread : 생산자 thread 가 만들어준 자원을 사용하는 Thread

발생할 수 있는 문제

→ 생산자 Thread 가 자원을 발생하지 않았는데  
소비자 자원을 사용하려할 때 발생하는 문제

→ Condition 클래스의 인스턴스를 이용해서 해결

3. Condition 클래스의 method

wait() : 현재 Thread 대기

notify() : 대기중인 Thread를 다시 수행

## < Semaphore >

: 동기화 클래스

. 생성할 때 동시에 수행되는 Thread 의 개수를 받아서 생성

사용하는 method 는 Lock 과 동일

acquire()

: 사용할 수 있는 자원의 개수를 1개 줄이고 작업을 수행

release()

: 사용할 수 있는 자원의 개수를 1개 증가시키고 작업을 수행

# **\*\* NetWork**

## **< Socket >**

: 네트워크 Interface 카드를 추상화한 클래스 또는 객체  
(통신을 위한 SoftWare tool)

### **통신방식**

#### **→ socket 의 직접 사용 여부**

: 직접 생성해서 통신하는 방식 .  
저수준 통신 . 효율은 좋지만 구현이 어려움  
고수준으로 Wrapping 된 클래스나 객체를 이용하는 방식 : URL 통신이 대표적

#### **→ 연결 방식에 따른 분류**

##### **TCP 통신**

: 송신 측과 수신측이 연결된 상태에서 대화를 주고받는 식으로 데이터를 전송

##### **UDP 통신**

: 송신 측이 수신측으로 일방적으로 데이터를 전송  
- 효율은 좋지만 데이터를 받았다는 사실을 알 수 없다 .  
\* SMART Phone 의 Push 가 이 방식으로 구현

#### **→ 연결하는 대상에 따른 분류**

##### **UNICAST**

→ 1 : 1 통신 // Q & A

##### **MULTICAST**

→ 1 : Group // 화상회의

##### **BROADCAST**

→ 1 : 전체 // 방송

##### **ANYCAST**

→ 아무나

## < 자신의 컴퓨터 주소 >

localhost  
: 내부에서 접속

→ 127.0.0.1  
→ 0:: 0:: 0:: 0:: 0:: 0:: 0:: 1

내부에서 접속하지만 외부로 나갔다가 들어오는 효과  
방화벽이 열려 있어야 한다

## \*\* 외부 데이터 사용

### < 파일 처리 >

#### 과정

- open 함수를 이용해서 파일을 개방
- read 나 write 함수를 이용해서 파일을 사용
- close 함수를 이용해서 파일과의 연결을 해제

#### 파일 기록

- Open 함수에 파일 경로 와 'w' 그리고 인코딩 방식 3개의 parameter 설정해서 파일을 개방
- Encoding 방식을 생략하면 현재 encoding 방식이 적용
- open 을 이용해서 개방한 파일 객체에  
write(writelines) 함수를 이용해서 기록 가능

#### 파일 읽기

- Open 파일 경로와  
'r' 그리고 Encoding 방식 3개의 parameter를 설정해서 파일을 개방  
Encoding 방식을 생략하면 현재 Encoding 방식이 적용
- read 함수를 이용하면 파일의 내용 전체를 읽어오고  
readline 을 이용하면 줄 단위로 읽을 수 있고  
readlines를 이용해서 줄단위로 끊어서 list 로 읽어올 수 있다
- read(Byte 크기) : 바이트 크기 만큼 읽어 옴

## < open 함수의 두번째 parameter >

r

: 읽기

x

: 베타적 생성 → 파일이 존재하면 error

a

: 파일에 기록을 하는데 존재하는 경우에는 맨 뒤에 이어 붙이기

b

: 바이너리 모드 → 바이트 단위로 읽을 때

t

: 텍스트 모드

+

: 읽기와 쓰기 모두 가능한 형태로 개발

## < 경로 >

: 절대 경로

### 루트로부터의 경로

디렉토리 구분 기호는 windows 는 ₩ 그 이외의 경우는 /

### → 상대 경로

현재 위치로부터의 경로

../

: 상위 디렉토리

./

: 현재 디렉토리. ./ 를 생략하고 디렉토리 이름부터 시작해도 현재 디렉토리

/

: web 에서 사용하는 방식 Server로의 root로 부터의 경로

- 일반적으로 프로토콜 ://IP. port번호 /

```

print("File Handling -----")
print("파일에 읽고 쓰기 ")
# 현재 디렉토리 확인
import os

print(os.getcwd())

try:
    #file 을 쓰기 모드로 개방
    # 'a' --> 추가
    # 'w' --> 기록만
    file = open('./test.txt', 'w')
    # file 에 기록
    file.write("Hello FileHandling")
    file.write("#n")
    li = ["I", "wanna", "be", "a", "richMan",
          "and", "being", "a", "irreplaceable", "person", "할 수 있다."]
    # list 의 내용을 #n 을 중간에
    file.write("\n".join(li))

    file.close()
except Exception as e:
    print(e)

```

```

----- 공유자원 수정 문제 발생
Thread Class 상속 ---- Thread Class 생성
id=Thread-1 증가하기 전 ---> 0
File Handling -----
파일에 읽고 쓰기

```

Project ▾

- pythonProject ~/PycharmProjects/py
  - pdf
  - venv
  - 0126\_2nd.py
  - 0127\_3rd.py
  - 0128\_4th.py
  - 0203\_5th.py
  - 0204\_6th.py
  - main.py
  - map.html
  - test.txt

0204\_6th.py × test.txt ×

```

1 Hello FileHandlingWnI
2
3 wanna
4
5 be
6
7 a
8
9 richMan
10
11 and
12
13 being
14
15 a
16
17 irreplaceable
18
19 person
20
21 할 수 있다.

```

```

print("파일을 읽기 모드로 개방 -----")
import time
try:
    # 파일을 읽기 모드로 개방
    file = open('./test.txt', 'r')

# 전체 읽어오기 --> 용량이 크면 메모리 부족으로 읽어오지 못함
# print(file.read())

# 줄 단위 읽기
    for line in file:
        print(line)

        time.sleep(2)

    file.close()
except Exception as e:
    print(e)

```

파일을 읽기 모드로 개방 -----

Hello FileHandlingWnI

wanna

be

a

richMan

and

being

a

irreplaceable

person

할 수 있다.

들여쓰기 조심하자.

## 〈 파일 입출력 간소화 〉

with open() as 별명:

별명을 이용한 읽기와 쓰기

- open에 인해서 열린 파일을 close 하지 않아도  
예외 발생 여부에 상관없이 마지막에 close 를 해준다.

## 〈 CSV(Comma Separated Values) 〉

- 구분자로 구분된 텍스트 파일
- 공백이나 쉼표 또는 탭이나 엔터로 구분된 문자열
- 변하지 않는 과거의 데이터를 제공할 때 많이 사용

Python에서는

csv 와 같은 모듈을 사용해서 직접 자료구조를 설정

pandas 와 같은 package 에서는

바로 읽어서 자신이 사용하는 자료구조(DataFrame)으로 바로 변환

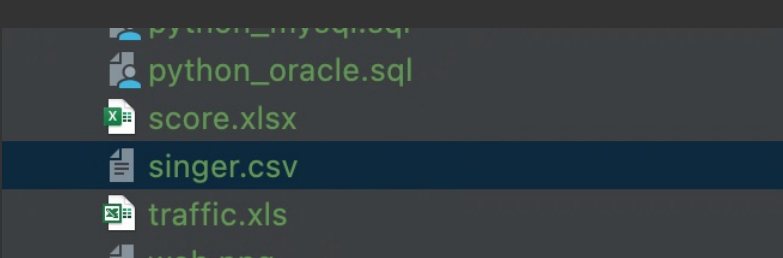
→ csv 파일을 읽을 때 확인할 내용

구분자

헤더의 여부(제목 셀의 존재 여부)

한글 포함 여부 → Encoding 설정





	A	B	C	D	
1	1	태연	1989.3.9	소녀시대	
2	2	재경	1988.12.24	레인보우	
3	3	아이린	1991.3.29	레드벨벳	
4					

```
print("CSV 파일 핸들링 -----")
# 파일 읽기를 이용해서 csv 파일 읽기 ---> 모든 데이터를 분할해서 list 로 생성
import csv
with open("./data/singer.csv", 'r') as f:
    # 전체 읽기
    content = f.read()

    # 구분자 단위로 분할
    data = content.split('.')

    # 자료형 확인
    print(type(data))

    # 전체 출력
    for imsi in data:
        print(data)

# CSV 모듈을 이용해서 읽기 csv.reader(file 객체)를 분할해서 list 로 생성
import csv
#한글이 깨지면 Encoding 옵션에 인코딩 방식을 설정
with open("./data/singer.csv", 'r', encoding="utf-8") as f:
    # csv 로 읽기
    # 구분자가 , 가 아니면 delimiter 에 구분자를 설정해주면 된다.

    data = csv.reader(f)

    # 자료형 확인
    print(type(data))

    # 전체 출력
    for imsi in data:
        print(imsi)

print("기록하기 -----")
import csv
# newLine 옵션을이용해서 빈줄이 생기지 않도록 설정
# 한글이 깨지면 Encoding option에 Encoding 방식을 서릿
with open("./data/singer.csv", 'a', newline='') as f:
    # csv로 기록
    wr = csv.writer(f)
    wr.writerow([4, "karina", "2000-01-01", "aespa"])
```

```
CSV 파일 핸들링 -----
<class 'list'>
['1,태연,1989-03-09,소녀시대\n2,재경,1988-12-24,레인보우\n3,아이린,1991-03-29,레드벨벳\n']
<class '_csv.reader'>
['1', '태연', '1989-03-09', '소녀시대']
['2', '재경', '1988-12-24', '레인보우']
['3', '아이린', '1991-03-29', '레드벨벳']
기록하기 -----
```

Plugins supporting *.csv files found.	
1	1,태연,1989-03-09,소녀시대
2	2,재경,1988-12-24,레인보우
3	3,아이린,1991-03-29,레드벨벳
4	4,karina,2000-01-01,aespa
5	

## < 파이썬의 내장 모듈인 CSV 모듈을 이용 >

csv.reader( 파일 객체 )

: 구분자 단위로 분할해서 list 로 생성  
줄 바꿈이 있으면 새로운 list 생성

→ csv.writer( 파일 객체 )

: 사용하여 쓰기모드를 만들고, 만들어진 객체의 writerrow를 이용해서 행단위 기록

많은 양의 데이터 중

일부분을 가지고 데이터 분석을 하는 경우

분석에 사용하는 데이터만 csv 파일로 기록해서 재사용하는 것이 효율적

## < Binary File >

: 바이트 단위로 파일에 읽고 쓰기

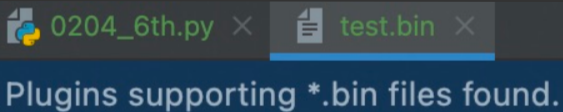
text는 문자열로 바로 기록하고 읽을 수 있다 .

그 이외의 내용은 텍스트 단위로 읽고 쓸수가 없다 → 이런경우 바이트 단위로 읽고 써야한다.

. wb 나 rb 모드로 파일을 개방하고 읽고 쓰는 method는  
read 와 write를 사용하는데 bytes 단위로 읽고 쓸 수 있다

```
print("Byte 단위로 파일을 읽고 쓰기-----")
with open('./data/test.bin', 'wb') as f:
    f.write("Hello 커피".encode())

with open('./data/test.bin', 'rb') as f:
    # print(f.read())
    print(f.read().decode())
```



0204\_6th.py × test.bin ×

Plugins supporting \*.bin files found.

1 Hello 커피



```
Byte 단위로 파일을 읽고 쓰기-----
Hello 커피
```

# < Serializable(직렬화) >

:객체 단위로 파일에 읽고 쓰는 것

## 응용프로그램을 만들거나

데이터를 주고 받는 통신 시스템에서 많이 사용  
파이썬에서는 pickle 모듈을 이용해서 직렬화 가능

→ 저장 ( 데이터 백업시 Dump 한다는 표현을 사용)

`pickle.dump( 출력할 객체 , 파일 객체 )`

→ 읽기

`pickle.load( 파일 객체 )` - 1개를 리턴

`pickle.loads( 파일 객체 )` - 전체를 바이트 단위로 읽기

\*→ 별로 좋지 않음 데이터는 하나를 만들어서 하나를 기록하는게 좋다

```
class Vo:
    def __init__(self):
        self.num = None

    def setNum(self, num):
        self.num = num

    def setName(self, name):
        self.name = name

    def getNum(self):
        return self.num

    def getName(self):
        return self.name

    def toString(self):
        return f"번호:{self.num} 이름:{self.name}"

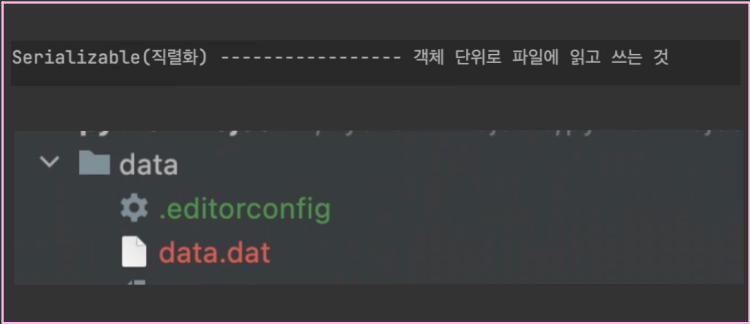
# 직렬화 할 인스턴스 생성
dto1 =Vo()
dto1.setNum(1)
dto1.setName("Nani")

dto2 =Vo()
dto2.setNum(2)
dto2.setName("SInde_Eru")

li = [dto1, dto2]

import pickle
with open('./data/data.dat', 'wb') as f:
    pickle.dump(li, f)

import pickle
with open('./data/data.dat', 'rb') as f:
    result = pickle.load(f)
    for Vo in result:
        print(Vo.toString())
```



Serializable(직렬화) ----- 객체 단위로 파일에 읽고 쓰는 것  
번호:1 이름:Nani  
번호:2 이름:SInde\_Eru

## < 파일 압축 압축 해제 >

→ zipfile 이라는 모듈을 사용

### 압축

: ZipFile 이라는 함수로 압축할 객체를 생성  
write 함수를 이용해서 압축

### 압축 해제

: 압축된 파일을 가지고 ZipFile 이라는 함수로 객체를 만들고  
extractall 함수 호출

→ tar

: unix , Linux 압축 파일형식

### tarfile 이라는 모듈 사용

#### open 함수를 이용해서

압축할 객체를 생성하고

add 함수를 이용해서 파일을 추가

### 압축해제

→ 압축된 파일을 가지고 extractall 함수 호출

# < 엑셀 파일 읽고 쓰기 > —> Openpyxl package 사용

: 엑셀 파일을 읽고 쓰기 위한 package 가 있는데

데이터 분석을 할 때는 pandas package에 Excel 파일을 읽어서  
DataFrame 으로 변환해주는 API 가 있기 때문에 어떤 용도로 하느냐 따라서  
읽고 쓰는 방법이 달라진다 .

Openpyxl package 는 python 의 기본 package 가 아님

—> 설치 : 터미널에 작성

`pip install openpyxl`

—> 작업단위

File(Work Book)

—> Sheet

—> Row(Column)

—> Cell

	A	B	C	D	E	
1	name	국어	영어	수학		
2	주시현	89	93	98		
3	최경우	95	87	73		
4	정정원	85	89	79		
5						

```
print("excel 파일 읽기 -----")
import openpyxl

# 엑셀 파일 포인터
wb = None

try:
    #엑셀 파일에 포인터를 생성
    wb = openpyxl.load_workbook('./data/score.xlsx')

    # 엑셀 파일 경로 확인
    print(wb)

    # sheet 가져오기
    # 현재 활성화된 sheet를 가져온다
    ws = wb.active

    # 여러개의 sheet를 가져올 때
    # ws = wb.get_sheet_by_name("sheet이름")

    # 행단위 접근 --> 각 행을 튜플로 접근
    # 튜플로 접근하는 이유
    # 접근 시 고치지 못하게 하기 위해서 --> 읽어오기 위함임으로
    for row in ws.rows:
        print(row)
        # 각행의 index
        print("Horizon-----")
    for row in ws.rows:
        # 행의 첫번째 데이터
        print(row[0].value)
    print("Horizon-----")
    for row in ws.rows:
        # 각행의 인덱스 추출
        print(row[0].row)

except Exception as e:
    print(e)

finally:
    if wb != None:
        wb.close()
```

```
excel 파일 읽기 -----
<openpyxl.workbook.workbook.Workbook object at 0x7f8ed07249d0>
(<Cell 'sheet1'.A1>, <Cell 'sheet1'.B1>, <Cell 'sheet1'.C1>, <Cell 'sheet1'.D1>)
(<Cell 'sheet1'.A2>, <Cell 'sheet1'.B2>, <Cell 'sheet1'.C2>, <Cell 'sheet1'.D2>)
(<Cell 'sheet1'.A3>, <Cell 'sheet1'.B3>, <Cell 'sheet1'.C3>, <Cell 'sheet1'.D3>)
(<Cell 'sheet1'.A4>, <Cell 'sheet1'.B4>, <Cell 'sheet1'.C4>, <Cell 'sheet1'.D4>)
Horizon-----
name
주시현
최경우
정정원
Horizon-----
1
2
3
4
```

# < 데이터 베이스 사용 >

-> Programming 안에서 데이터 베이스 연동

1. 데이터 베이스 interface(driver, package)를 이용해서 직접 데이터를 읽고 쓰는 방법

2. 데이터 베이스 연동 FrameWork를 이용하는 방식

- SQL Mapper

: SQL 을 별도로 작성하고 파라미터나 Mapping이나 결과 Mapping 을 자동 수행

- ORM

: SQL 대신에 설정을 이용해서 Instance 와 Database Table 행 Mapping  
Django ORM , Pony ORM, peewee 등이 있다

## MYBATIS , HIBERNATE 차이 AOP , IOC , ORM

-> 데이터베이스 연동 시 준비

1. 데이터베이스 interface에 해당하는 package 를 설치

2. 단 sqlite3 Interface python이 내장

-> pyMySQL(pip install pyMySQL)

. 데이터베이스경로 (url) : DOMAIN 이나 IP + Port 번호

. 데이터베이스이름

( oracle 은 sid 나 servicename 이라고 하고 다른 데이터베이스는 대부분 name)

. 계정 과 비밀번호

. encoding 방식

```
print("mysql 접속 -----")
import pymysql

con = pymysql.connect(host='127.0.0.1', port=3306,
                      user='singsiuk', password='ssw0304',
                      db='python', charset='utf8')

print(con)
```

```
mysql 접속 -----
<pymysql.connections.Connection object at 0x7fa9282ebc10>
```

Statistics 1 x Output

create database python | Enter a SQL expression to filter

Name	Value
Updated Rows	1
Query	create database python
Finish time	Fri Feb 04 16:15:11 KST 2022