

computational thinking



# 파이썬 프로그래밍



## 12장 파일과 예외처리

# 파일의 필요성

- 프로그램에서 만든 데이터를 영구히 저장
  - 하드 디스크에 파일 형태로 저장



메모리

VS

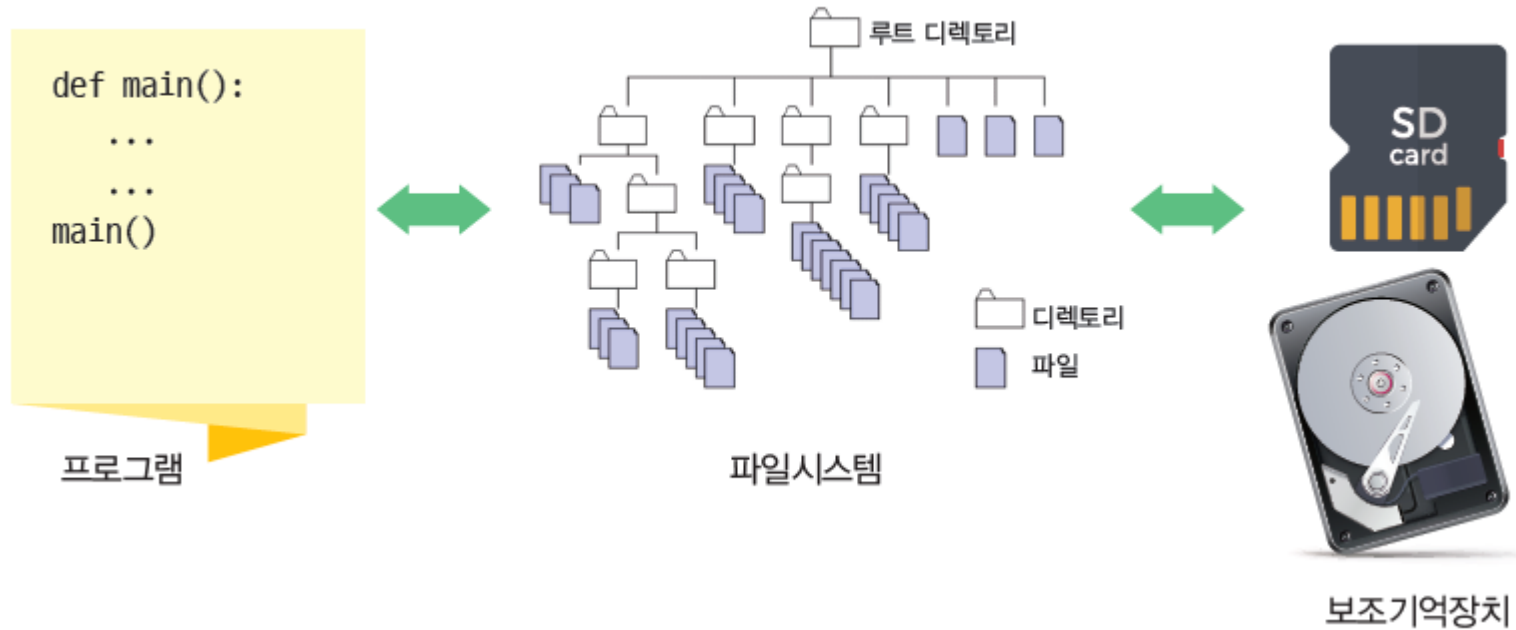


SSD, 하드 디스크

# 파일의 개념

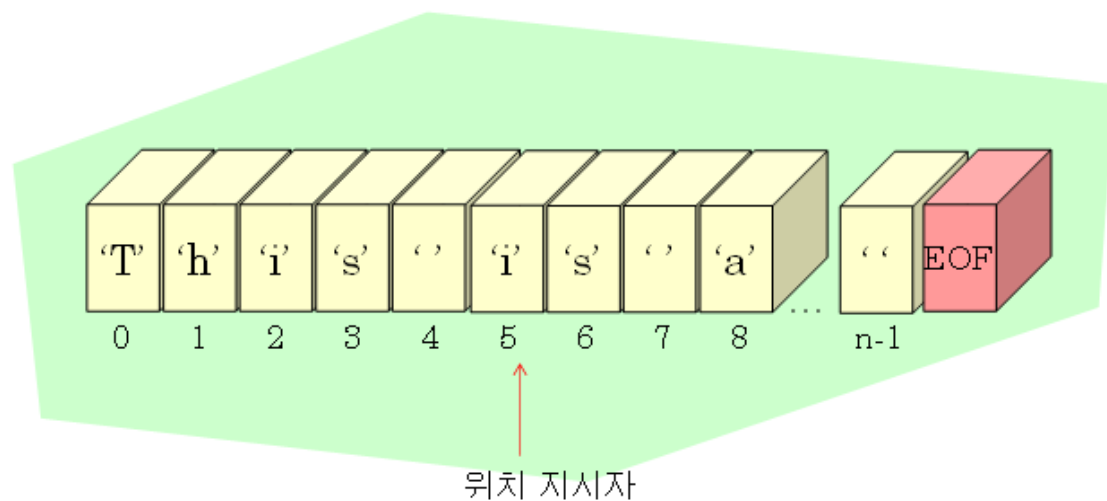
## ■ 파일(file)

- 보조 기억장치에 문서, 소리, 그림, 동영상과 같은 자료를 모아 놓은 것
- 보조기억장치에서 논리적인 정보 단위
- 운영체제는 파일 조작에 관련된 기능을 라이브러리로 제공



# 파일의 논리적인 구조

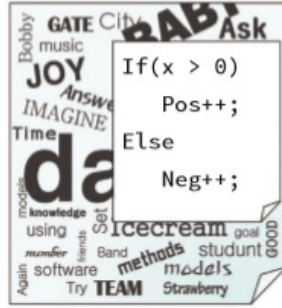
- 파일 안에는 바이트들이 순차적으로 저장
  - 파일의 끝에는 EOF(end-of-file) 마커가 있음
- 위치 표시자(position indicator)
  - 파일의 입출력 동작이 발생하는 위치를 나타냄



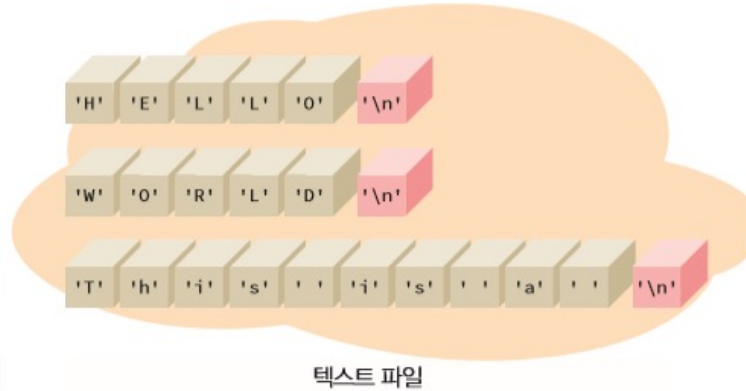
파일의 논리적인 구성

# 파일의 종류

## ■ 텍스트 파일(text file)



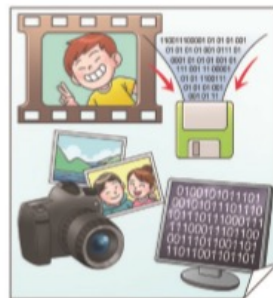
텍스트 파일: 문자로 구성된 파일



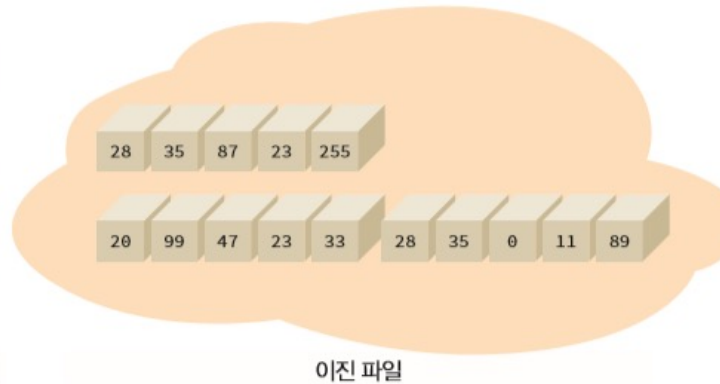
텍스트 파일

## ■ 이진 파일(binary file)

- 사진, 동영상, 음악 파일 등



이진파일: 데이터로 구성된 파일



이진 파일

# 파일 열고 닫기

## ■ 파일 열기

- 파일객체 = `open`(파일이름, 파일모드, `encoding='utf-8'`)
  - 파일 객체를 리턴
  - `encoding='utf-8'`: Python의 기본 encoding
  - `encoding='euc-kr'`: Windows에서 한글이 포함된 파일 (저장 시 utf-8로 저장 가능)

## ■ 파일 닫기: 파일객체.`close()`

Syntax: 함수 정의

**형식**    파일객체 = `open`(파일이름, 파일모드)  
          파일객체.`close()`

**예**    `infile = open("input.txt", "r")`  
      `...`  
      `infile.close()`

파일 객체      파일의 이름(name)      파일을 여는 모드(mode)

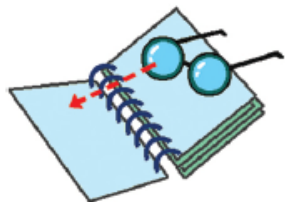
파일을 연다  
파일객체=open()

파일에 데이터를  
읽거나 쓴다

파일을 닫는다  
파일객체.close()

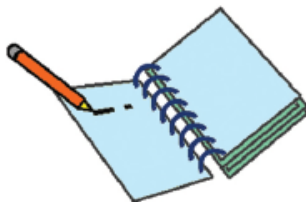
# 파일 모드

파일 모드	모드 이름	설명
"r"	읽기 모드(read mode)	<ul style="list-style-type: none"> <li>파일의 처음부터 읽음</li> </ul>
"w"	쓰기 모드(write mode)	<ul style="list-style-type: none"> <li>파일의 처음부터 저장</li> <li>파일이 없으면 생성</li> <li>파일이 이미 존재하면 기존 내용은 지워짐</li> </ul>
"a"	추가 모드(append mode)	<ul style="list-style-type: none"> <li>파일의 끝에 저장</li> <li>파일이 없으면 생성</li> </ul>
"r+"	읽기와 쓰기 모드	<ul style="list-style-type: none"> <li>파일을 읽고 쓸 수 있는 모드</li> <li>기존 파일은 그대로 두고 처음부터 덮어 쓰기 수행</li> </ul>
"w+"	읽기와 쓰기 모드	<ul style="list-style-type: none"> <li>파일을 읽고 쓸 수 있는 모드</li> <li>기존 파일에 있던 데이터는 완전히 지우고 새로 씀</li> </ul>



"r"

파일의 처음 부터 읽는다.



"w"

파일의 처음 부터 쓴다.  
만약 파일이 존재하면 기존의  
내용이 지워진다.



"a"

파일의 끝에 쓴다.  
파일이 없으면 생성 된다.

# 파일에서 읽기 #1

- 파일객체.read() 함수
  - 파일 전체 내용을 하나의 문자열로 읽어옴
  - 리턴값: 읽은 문자

<phones.txt>

```
홍길동 010-1234-5678  
김철수 010-1234-5679  
김영희 010-1234-5680
```

- 파일객체.read(num)
  - 파일에서 num개의 문자를 읽음
  - 리턴값: 읽어 들인 문자열

<file\_read01.py>

```
infile = open("phones.txt", "r", encoding='utf-8 ' ) # euc-kr: 한글  
s = infile.read(10)  
print(s);  
infile.close()
```

```
홍길동 010-12
```



# 파일에서 한 줄씩 읽기

- 파일객체.readline() 함수
  - 파일에서 한 줄을 읽음: new line('\n')까지 읽음

<file\_readline01.py>

```
infile = open("phones.txt", "r", encoding='utf-8')
s = infile.readline()
print(s)

s = infile.readline()
print(s)

s = infile.readline()
print(s)

infile.close()
```

홍길동 010-1234-5678

김철수 010-1234-5679

김영희 010-1234-5680

# 파일 전체 읽기

## ■ read() 함수

- 파일이 큰 경우 많은 양의 메모리를 필요함: 권장 방법은 아님

```
infile = open("phones.txt", "r")
s = infile.read()
print(s)
infile.close()
```

## ■ readlines() 함수

- 각 라인이 저장된 리스트를 반환

```
infile = open("phones.txt", "r")
lines = infile.readlines()
print(lines)
for line in lines:
    print(line)
infile.close()
```

```
['홍길동 010-1234-5678\n', '김철수 010-1234-5679\n', '김영희 010-1234-5680\n']
홍길동 010-1234-5678

김철수 010-1234-5679

김영희 010-1234-5680
```

# 파일의 끝까지 한 줄씩 읽기

- 파일의 끝(End of File: EOF) 구분
  - EOF를 만나면, `empty string('')`을 리턴함

<file\_readline02.py>

```
infile = open("phones.txt", "r", encoding='utf-8')
line = infile.readline()

while line != "": # EOF가 아닐 때까지 반복
    print(line, end='')
    line = infile.readline()

infile.close()
```

```
홍길동 010-1234-5678
김철수 010-1234-5679
김영희 010-1234-5680
```

# 파일을 읽고 공백 문자 제거

- `str.rstrip()` 함수
  - 문자열의 오른쪽 공백 문자를 제거하는 메소드
  - 파일에서 줄 **마지막에 붙은 '\n' 을 제거할 때** 유용함

<file\_readline03.py>

```
infile = open("phones.txt", "r")
```

```
for line in infile:  
    line = line.rstrip()  
    print(line)
```

파일 객체(infile)에서  
한 라인씩 읽어옴

```
infile.close()
```

```
홍길동 010-1234-5678  
김철수 010-1234-5679  
김영희 010-1234-5680
```

# 파일 문자 단위로 읽기

- 파일에서 문자 단위로 읽기
  - read(1): 한 문자씩 읽음

<file\_read\_char.py>

```
infile = open("input.txt", "r")
ch = " "
while ch != "" :
    ch = infile.read(1)
    print(ch, end='')

infile.close()
```

```
홍길동
김철수
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
```

<input.txt>

```
홍길동
김철수
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
```

# 파일에 데이터 쓰기 #1

- 파일열기: 쓰기 모드
  - 파일객체 = open(파일이름, "w")
- 파일객체.write()

<file\_write01.py>

```
outfile = open("phones1.txt", "w")
```

```
outfile.write("홍길동 010-1234-5678")  
outfile.write("김철수 010-1234-5679")  
outfile.write("김영희 010-1234-5680")
```

new line('\n')이 없기  
때문에 1줄로 저장됨

```
outfile.close()
```

- phones1.txt 파일 내용

```
홍길동 010-1234-5678김철수 010-1234-5679김영희 010-1234-5680
```

## 파일에 데이터 쓰기 #2

- 파일에 데이터를 쓰기 전에 동일한 파일이 있는지 확인
  - `os.path.isfile(파일이름)`

<file\_write02.py>

```
import os.path

filename = "phones1.txt"

if os.path.isfile(filename):
    print(f"{filename} 파일이 이미 존재합니다. ")
else:
    outfile = open("phones1.txt", "w")
    outfile.write("홍길동 010-1234-5678")
    outfile.write("김철수 010-1234-5679")
    outfile.write("김영희 010-1234-5680")
    outfile.close()
```

동일한 이름의 파일이 이미 존재합니다.

# 파일 닫기

- 일반적인 방법

```
f = open("test.txt", "w")  
# 파일 작업 수행  
  
f.close() # 파일 닫음
```

<file\_close01.py>

- 예외 처리: finally 내부

```
try:  
    f = open("test.txt", "w")  
    # 파일 작업 수행  
except:  
    # 예외 처리 문장  
    print('exception 발생')  
finally:  
    # 예외가 발생하더라도 반드시 실행  
    f.close()
```

- with 구문

```
with open("test.txt", "w") as f:  
    f.write("Hello")  
  
# with 블록을 빠져 나오면 자동으로 파일이 닫힘
```



# Lab: 매출 파일 처리

- sales.txt 파일 생성

입력 파일에는 상점의 하루 매출이 한 줄에 정수로 기록되어 있다. 예를 들면 다음과 같다.

*sales.txt*

```
1000000  
1000000  
1000000  
500000  
1500000
```

출력 파일은 다음과 같아야 한다.

*summary.txt*

```
총매출 = 5000000  
평균 일매출 = 1000000.0
```

# Solution: sales.py

```
# 입력 파일 이름과 출력 파일 이름을 입력
infilename = input("입력 파일 이름: ");
outfilename = input("출력 파일 이름: ");

# 입력과 출력을 위한 파일을 연다.
infile = open(infilename, "r")
outfile = open(outfilename, "w")

# 합계와 횟수를 위한 변수를 정의한다.
sum = 0
count = 0

# 입력 파일에서 한 줄을 읽어서 합계를 계산한다.
for line in infile:
    dailySale = int(line)
    sum = sum + dailySale
    count = count + 1

# 총매출과 일평균 매출을 출력 파일에 기록한다.
outfile.write("총매출 = "+ str(sum)+"\n")
outfile.write("평균 일매출 = "+ str(sum/count))

infile.close()
outfile.close()
```

## 실행 결과

입력 파일 이름: sales.txt  
출력 파일 이름: summary.txt

## summary.txt 파일 내용

총매출 = 5000000  
평균 일매출 = 1000000.0

# 텍스트 입출력 기법

- 데이터 추가하기
  - append 모드: “a”
    - 기존 파일의 마지막에 추가

<file\_append.py>

```
outfile = open("phones.txt", "a")

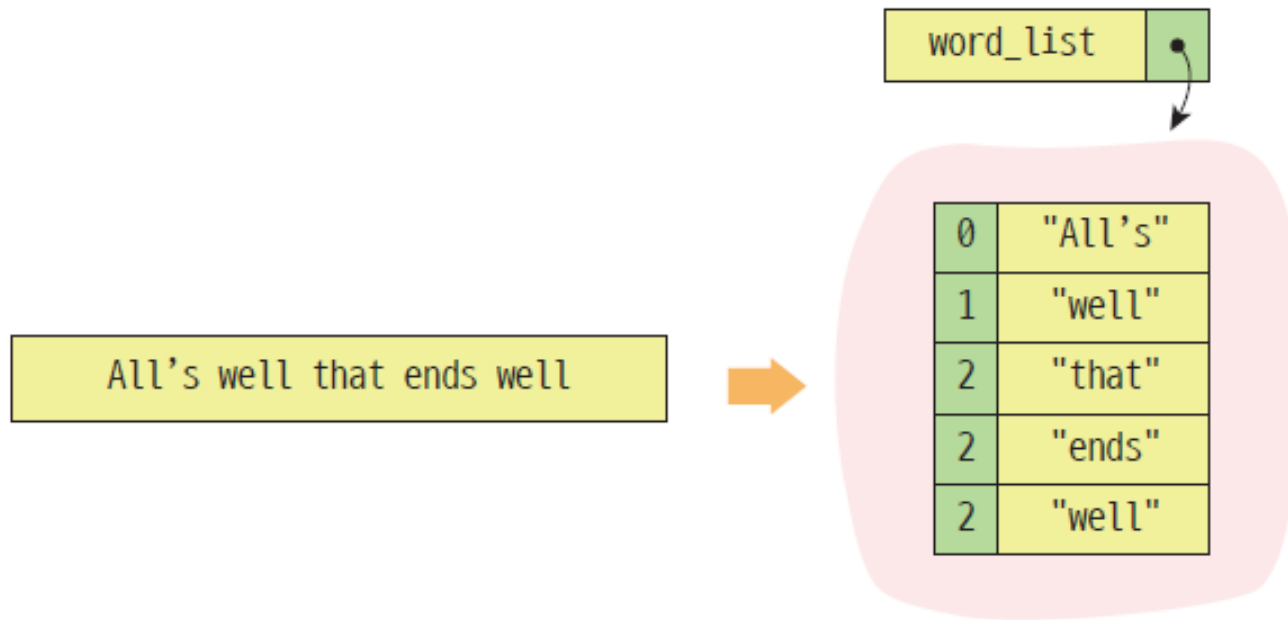
outfile.write("최무선 010-1111-2222\n")
outfile.write("정중부 010-2222-3333\n")

outfile.close()
```

```
홍길동 010-1234-5678
김철수 010-1234-5679
김영희 010-1234-5680
최무선 010-1111-2222
정중부 010-2222-3333
```

# 텍스트 입출력 기법

- 파일에서 단어 읽기
  - 파일의 내용을 공백을 기준으로 분리



# 파일을 읽어서 단어로 분리하기

- `str.split(분리문자)`
  - 문자열을 공백 기준으로 분리하고 분리된 문자들의 리스트를 리턴

<word\_split.py>

```
infile = open("proverbs.txt", "r")

for line in infile:
    line = line.rstrip() # 오른쪽 '\n' 제거
    word_list = line.split() # 공백기준으로 분리
    for word in word_list:
        print(word);

infile.close()
```

<proverbs.txt>

```
All's well that ends well.
Bad news travels fast.
Well begun is half done.
Birds of a feather flock together.
```

```
All's
well
that
ends
well.
Bad
news
travels
fast.
Well
begun
is
half
done.
Birds
of
a
feather
flock
together.
```

## Lab: 각 문자 횟수 세기

- 파일 안의 각 문자들이 몇 번이나 나타나는지를 세는 프로그램을 작성하자.

```
filename = input("파일명을 입력하세요: ").strip()
infile = open(filename, "r")

freqs = {} # 딕셔너리 생성

# 파일의 각 줄에서 문자를 추출한 다음 각 문자를 dict에 추가
for line in infile:
    for char in line.strip():
        if char in freqs:
            freqs[char] += 1 # 딕셔너리에 key(char)가 있으면 증가
        else:
            freqs[char] = 1 # 딕셔너리에 key(char)가 없으면 추가
print(freqs)
infile.close()
```

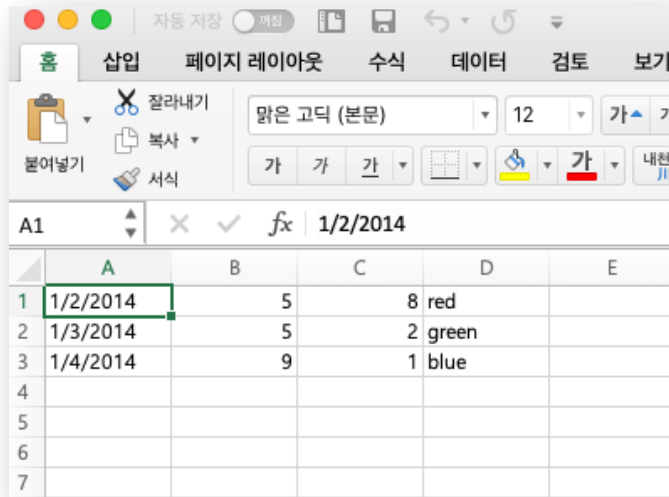
<count\_letter.py>

```
파일명을 입력하세요: proverbs.txt
{'A': 1, 'l': 11, '"': 1, 's': 7, ' ': 16, 'w': 3, 'e': 12, 't': 7, 'h': 4, 'a': 7, 'n': 4, 'd': 4, '.': 4, 'B': 2, 'r': 4, 'v': 1, 'f': 5, 'W': 1, 'b': 1, 'g': 2, 'u': 1, 'i': 2, 'o': 4, 'c': 1, 'k': 1}
```

# Lab: CSV 파일 읽기

## ■ CSV(Comma Separated Values) 형식

- 콤마(‘,’)로 각 값들을 구분한 텍스트 파일 형식
- 엑셀과 같은 스프레드시트나 데이터베이스에서 가장 널리 사용되는 입출력 형식
  - 엑셀 또는 메모장에서 csv 파일을 확인 가능
- 파이썬은 CSV 형식을 읽기 위해서 **csv 모듈**을 제공
  - 이 모듈을 이용하면 CSV 파일을 쉽게 읽을 수 있음



The screenshot shows a spreadsheet application window with a menu bar (홈, 삽입, 레이아웃, 수식, 데이터, 검토, 보기) and a toolbar. The active cell is A1, containing the date '1/2/2014'. The spreadsheet contains the following data:

	A	B	C	D	E
1	1/2/2014	5	8	red	
2	1/3/2014	5	2	green	
3	1/4/2014	9	1	blue	
4					
5					
6					
7					

<data.csv>

```
1/2/2014,5,8,red  
1/3/2014,5,2,green  
1/4/2014,9,1,blue
```

# Solution

<csv\_reader01.csv>

```
f = open("data.csv", "r")

for line in f.readlines():
    line = line.strip() # 공백 문자를 없앴
    print(line)
    parts = line.split(",") # 한 줄의 데이터를 쉼표로 분리

    # 각 줄의 필드를 출력
    for part in parts:
        print("    ", part)
```

```
1/2/2014,5,8,red
    1/2/2014
    5
    8
    red
1/3/2014,5,2,green
    1/3/2014
    5
    2
    green
1/4/2014,9,1,blue
    1/4/2014
    9
    1
    blue
```

<data.csv>

```
1/2/2014,5,8,red
1/3/2014,5,2,green
1/4/2014,9,1,blue
```



# csv 모듈 사용 예제: csv.reader()

- csv.reader(파일객체)
  - iterator 타입의 reader 객체를 리턴

<csv\_reader02.csv>

```
import csv

f = open('data.csv', 'r')
csv_reader = csv.reader(f) # iterator 타입인 reader객체를 리턴
for line in csv_reader:
    print(line)

    for i in range(len(line)):
        print(f"[{i}]:", line[i])
    print()
f.close()
```

```
['1/2/2014', '5', '8', 'red']
[0]: 1/2/2014
[1]: 5
[2]: 8
[3]: red
['1/3/2014', '5', '2', 'green']
[0]: 1/3/2014
[1]: 5
[2]: 2
[3]: green
['1/4/2014', '9', '1', 'blue']
[0]: 1/4/2014
[1]: 9
[2]: 1
[3]: blue
```

## csv 파일에 저장: csv.writer()

- `writer = csv.writer(csvfile, 'w', newline='')`
  - csv파일을 쓰기 모드로 열고 `writer` 객체를 리턴함
  - `newline=''` :
    - Windows 경우 csv 모듈에서 데이터를 쓸 때, 각 라인 뒤에 빈 라인이 추가되는 문제 발생
- `writer.writerow(row)`
  - csv파일에 한 라인씩 저장함

<csv\_writer01.csv>

```
import csv

f = open('output.csv', 'w', encoding='utf-8', newline='')

writer = csv.writer(f)
writer.writerow([1, 'Alice', True])
writer.writerow([2, 'Bob', False])

f.close()
```

	A	B	C	D
1	1	Alice	TRUE	
2	2	Bob	FALSE	
3				
4				
5				
6				
7				
8				
9				

<output.csv>

# 디렉토리 작업

## ■ 디렉토리 관련 함수

- `os.getcwd()`: 현재 작업 디렉토리의 경로 얻기
  - cwd: current working directory
- `os.chdir(경로명)`: 작업 디렉토리 변경
- `os.listdir()`: 작업 디렉토리 안에 있는 파일들과 디렉토리 이름 반환

## ■ 파일 관련

- `os.path.isfile(파일이름)`: 파일 존재 여부 확인
- `os.path.isdir(디렉토리이름)`: 디렉토리 존재 여부 확인
- `str.endswith("확장자")`: 파일 확장자 검사

# 디렉토리 작업 예제

<dir01.csv>

```
import os

dir = os.getcwd()
print(dir)

subdir = "data"
os.chdir(subdir)
print(os.getcwd())

for filename in os.listdir():
    print(filename, end='')
    if os.path.isfile(filename):
        print(": 파일")
    elif os.path.isdir(filename):
        print(": 디렉토리")
```

```
/Users/changsu/workspace_swcoding/ch10_file
/Users/changsu/workspace_swcoding/ch10_file/data
proverbs.txt: 파일
sales.txt: 파일
output.csv: 파일
data.csv: 파일
input.txt: 파일
summary.txt: 파일
words.txt: 파일
```

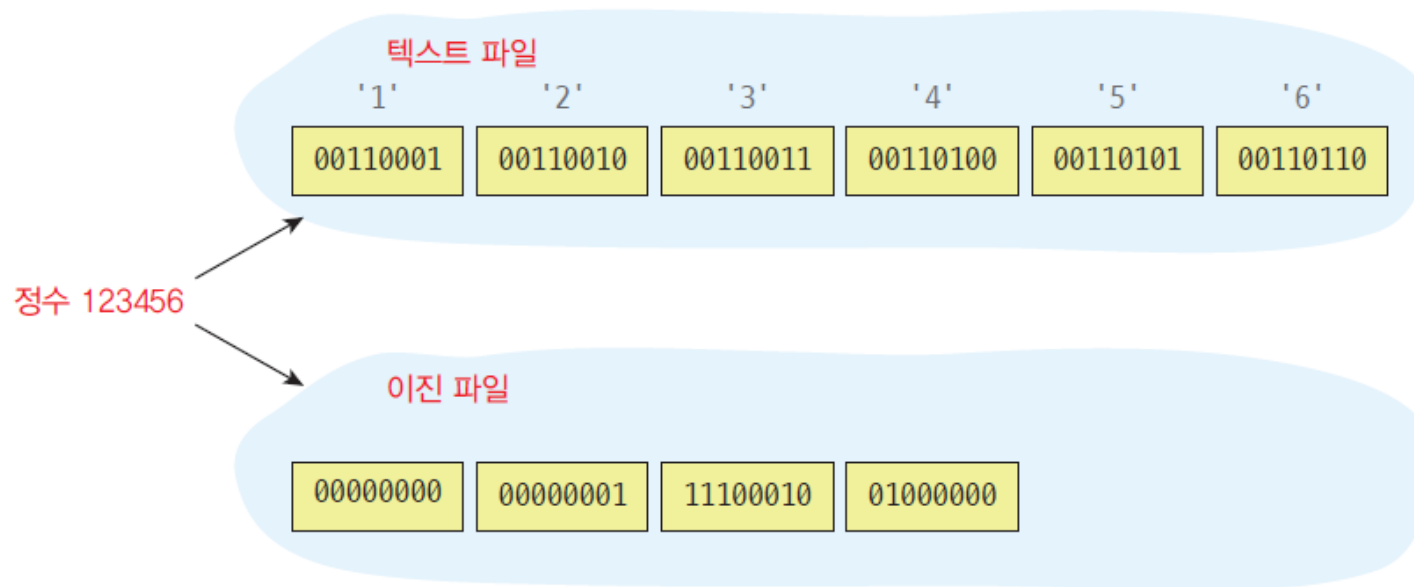
# 이진 파일

## ■ 이진 파일(binary file)

- 이진수 형태로 저장된 파일

## ■ 파일 열기

- 읽기 모드: `open(filename, "rb")`
- 쓰기 모드: `open(filename, "wb")`



# Lab: 이미지 파일 복사하기

- 하나의 이미지 파일을 다른 이미지 파일로 복사하는 프로그램을 작성하시오.



원본 파일 이름을 입력하시오: img1.jpg

복사 파일 이름을 입력하시오: abc.jpg

1024

...

1024

1024

1024

771

0

len(copy\_buffer)의 값이 0  
=> 파일의 끝

img1.jpg를 abc.jpg로 복사하였습니다.

# Solution

<binary\_filecopy.py>

```
filename1 = input("원본 파일 이름을 입력하시오: ")
filename2 = input("복사 파일 이름을 입력하시오: ")

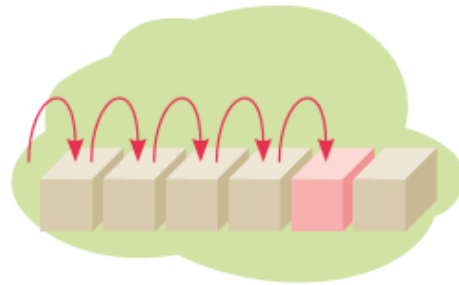
infile = open(filename1, "rb")
outfile = open(filename2, "wb")

# 입력 파일에서 1024 바이트씩 읽어서 출력 파일에 쓴다.
# 파일의 마지막 부분에서는 읽어 들인 바이트 수만큼 파일에 저장
while True:
    copy_buffer = infile.read(1024)
    print(len(copy_buffer)) # 실제 읽어온 바이트 수 출력
    if not copy_buffer:    # 파일의 끝인 경우, empty byte를 리턴
        break
    outfile.write(copy_buffer)

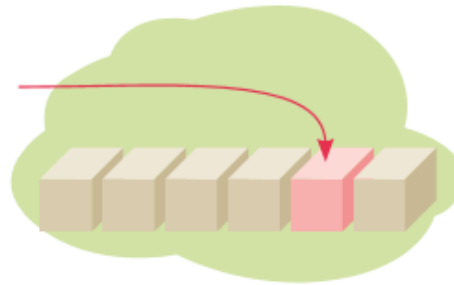
infile.close()
outfile.close()
print(filename1+"를 " + filename2 + "로 복사하였습니다. ")
```

# 임의 접근 파일

- 파일 포인터를 이동시켜서 랜덤하게 읽는다.

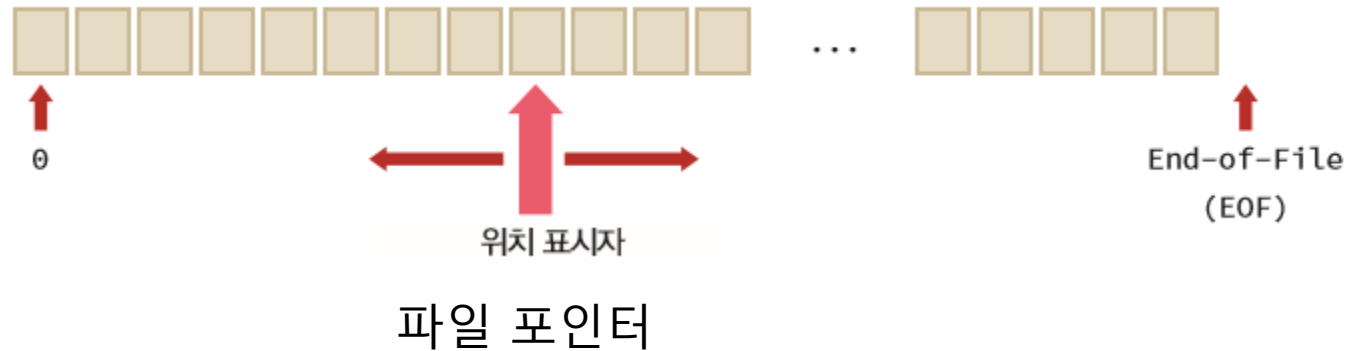


순차접근파일



임의접근파일

- 위치 표시자 (파일 포인터)의 이동





# 임의 접근

- `tell()`
  - 파일에서 현재의 위치를 리턴
- `seek(offset, whence=SEEK_SET)`
  - 임의의 파일 위치로 이동
  - `whence(기준 위치): 디폴트값 = SEEK_SET`
    - `SEEK_SET` or `0`: 파일의 시작 위치
    - `SEEK_CUR` or `1`: 파일의 현재 위치
    - `SEEK_END` or `2`: 파일의 끝

# 예제

<seek.py>

```
infile = open("proverbs.txt", "r+")
str = infile.read(10); # 파일 포인터가 10바이트 이동
print("읽은 문자열:", str)
position = infile.tell();
print("현재 위치:", position)

position = infile.seek(0, 0); # 파일의 처음으로 이동
str = infile.read(10);
print("다시 읽은 문자열:", str)
infile.close()
```

```
읽은 문자열 : All's well
현재 위치: 10
다시 읽은 문자열 : All's wel
```

# 객체 입출력(직렬화, 역직렬화)

- pickle 모듈 사용
  - dump()와 load() 메소드를 사용하여 바이트 형태로 객체를 쓰고 읽을 수 있음
- pickle.dump(obj, file)
  - obj 형태를 파일 객체(file)에 저장함
  - 객체를 바이트 스트림으로 변환해서 파일로 저장: 직렬화
- pickle.load(file)
  - 바이트 스트림이 객체 구조로 변환: 역직렬화
  - 한 줄씩 파일을 읽어오고 더 이상 로드할 데이터가 없으면 EOFError 발생
- 사용 형식

```
pickle.dump(데이터, 파일) # 객체를 파일로 저장  
  
객체 or 자료형 변수 = pickle.load(파일) # 파일에서 객체를 읽어옴
```

# 딕셔너리 파일 입출력 예제

<pickle\_dict.py>

```
import pickle

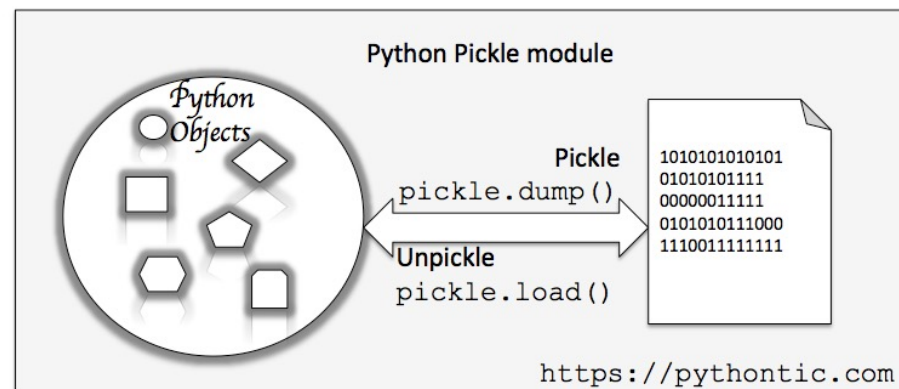
fout = open("pickle.bin", "wb")
mydict = dict()

mydict['a'] = 1
mydict['b'] = 10
mydict['c'] = 100

pickle.dump(mydict, fout)
fout.close()

fin = open("pickle.bin", "rb")
dict1 = pickle.load(fin)
fin.close()
print(dict1)
```

```
{'a': 1, 'b': 10, 'c': 100}
```



```
pickle.bin
1 8004 9517 0000 0000 0000 007d 9428 8c01
2 6194 4b01 8c01 6294 4b0a 8c01 6394 4b64
3 752e
```

# 객체 파일 입출력 예제

<pickle\_object.py>

```
import pickle
class Person(object):
    def __init__(self, id, name):
        self.id = id
        self.name = name
    def __repr__(self):
        return f"({self.id}, {self.name})"

p1 = Person(1, "Kim")
p2 = Person(2, "Park")
p3 = Person(3, "Lee")

person_list = [p1, p2, p3]
# 객체 리스트를 파일로 저장
fname = "person.pickle"
fout = open(fname, "wb")
pickle.dump(person_list, fout)
fout.close()

# 파일의 내용을 객체 리스트로 읽어옴
fin = open(fname, "rb")
plist = pickle.load(fin)
print(plist)
fin.close()
```

[(1, Kim), (2, Park), (3, Lee)]

<person.pickle>

	person.pickle	×
1	8004 9564 0000 0000 0000 005d 9428 8c08	
2	5f5f 6d61 696e 5f5f 948c 0650 6572 736f	
3	6e94 9394 2981 947d 9428 8c02 6964 944b	
4	018c 046e 616d 6594 8c03 4b69 6d94 7562	
5	6803 2981 947d 9428 6806 4b02 6807 8c04	
6	5061 726b 9475 6268 0329 8194 7d94 2868	
7	064b 0368 078c 034c 6565 9475 6265 2e	

# with 구문

## ■ with 구문

- 리소스의 액세스를 with 구문을 통해서 특정 블록 내부의 동작으로 제한
- 블록을 나가는 경우, 리소스의 해제 처리를 자동으로 보장

## ■ 파일 입출력에 with 구문 사용

- close() 메소드를 호출하지 않고 파일을 닫을 수 있음

```
with open('test.txt', 'w', encoding='utf-8') as f:
    f.write("with를 이용한 파일 저장 실습\n")
    f.write("newline 문자 사용.\n")
    f.write("파일 저장 완료\n")
```

```
with open('test.txt', 'r', encoding='utf-8') as f:
    for line in f:
        print(line.rstrip())
```

```
# 한번에 모든 라인 읽어오기, 리스트 형태로 반환함
with open('test.txt', 'r', encoding='utf-8') as f:
    lines = f.readlines()
    print(lines)
```

<with\_ex01.py>

```
with를 이용한 파일 저장 실습
newline 문자 사용.
파일 저장 완료
['with를 이용한 파일 저장 실습\n', 'newline 문자
사용.\n', '파일 저장 완료\n']
```

## with 구문 예제 #2

```
fname = "person.pickle"
# 객체 리스트를 파일로 저장
save_file = open(fname, "wb")
pickle.dump(person_list, save_file)
save_file.close()

read_file = open(fname, "rb")
plist = pickle.load(read_file)
print(plist)
read_file.close()
```



```
fname = "person.pickle"
# 객체 리스트를 파일로 저장
with open(fname, "wb") as save_file:
    pickle.dump(person_list, save_file)

with open(fname, "rb") as read_file:
    plist = pickle.load(read_file)
    print(plist)
```

파일 close() 부분이 없음

# 예외 처리

- 예외(Exception)
  - 오동작이나 결과에 악영향을 미칠 수 있는 **실행 중 발생한 오류**
- 예외의 종류
  - 잘못된 데이터 입력
  - 존재하지 않는 파일 열기
  - 정수를 0으로 나누는 경우
  - 리스트보다 큰 인덱스로 리스트의 원소를 접근하는 경우

```
(x, y) = (2, 0)
```

```
z = x / y
```

```
Traceback (most recent call last):
```

```
File "exception1.py", line 2, in <module>
```

```
z = x / y
```

```
ZeroDivisionError: division by zero
```





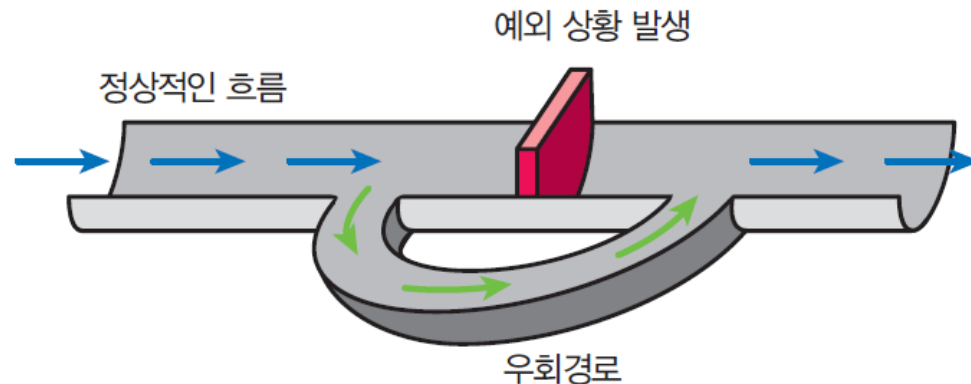
# 예외 처리의 개념

## ■ 예외 처리

- 오류가 발생 했을 때 오류를 사용자에게 알려줌
- 모든 데이터를 저장하게 한 후에 사용자가 프로그램을 종료할 수 있도록 하는 것

## ■ 예외 종류

- IOError: 파일을 열 수 없을 때 발생
- ImportError: 파이썬이 모듈을 찾을 수 없을 때 발생
- ValueError: 연산이나 내장함수에서 인수가 적절치 않은 값을 가질 때
- KeyboardInterrupt: 사용자가 인터럽트 키를 누를 때 발생
- EOFError: 내장 함수가 파일의 끝을 만나면 발생



# 파이썬 예외 처리

- try ~ catch 구조
  - try 블록
    - 예외가 발생할 가능성이 있는 문장을 추가
  - except 블록
    - 자신이 처리할 수 있는 예외의 종류 지정
    - 예외를 처리하기 위한 코드 추가

## Syntax: 예외 처리

**형식** try:  
    예외가 발생할 수 있는 문장  
except(오류):  
    예외를 처리하는 문장

**예** try:  
    z = x/y  
except ZeroDivisionError:  
    print ("0으로 나누는 예외")

예외가 발생할 수 있는 문장

예외

# 예외 처리 예제 #1

## ■ 예외 처리

- 프로그램이 비정상 종료되지 않고, 예외 처리 문장을 수행함

<exception01.py>

```
# 예외처리 #1
```

```
(x,y) = (2,0)
```

```
try:
```

```
    z = x/y
```

```
except ZeroDivisionError:
```

```
    print ("0으로 나누는 예외")
```

사용자 예외 메시지 출력

0으로 나누는 예외

```
(x,y) = (2,0)
```

```
try:
```

```
    z = x/y
```

```
except ZeroDivisionError as e:
```

```
    print(e)
```

파이썬이 제공하는 예외 메시지 출력

division by zero

## 예외 처리 예제 #2

### ■ ValueError 예외 처리

<exception02.py>

```
while True:
    try:
        n = input("숫자를 입력하시오 : ")
        n = int(n)
        break
    except ValueError:
        print("정수가 아닙니다. 다시 입력하시오. ")

print("정수 입력이 성공하였습니다!")
```

입력된 숫자가 정수가 아닌 경우 수행

```
숫자를 입력하시오 : 12.4
정수가 아닙니다. 다시 입력하시오.
숫자를 입력하시오 : 10
정수 입력이 성공하였습니다!
```

## 예외 처리 예제 #3

- IOError
  - 파일 오류 처리

<exception03.py>

```
try:
    fname = input("파일 이름을 입력하세요: ")
    infile = open(fname, "r")
except IOError:
    print("파일 " + fname + "을 발견할 수 없습니다.")
```

```
파일 이름을 입력하세요: 123.py
파일 123.py을 발견할 수 없습니다.
```

# 다중 예외 처리 구조

## 전체적인 구조



try :

예외가 발생할 수 있는 문장

except ExceptionI :

ExceptionI이면 이 블록이 실행된다.

except ExceptionII :

ExceptionII이면 이 블록이 실행된다.

else :

예외가 없는 경우에 실행된다.

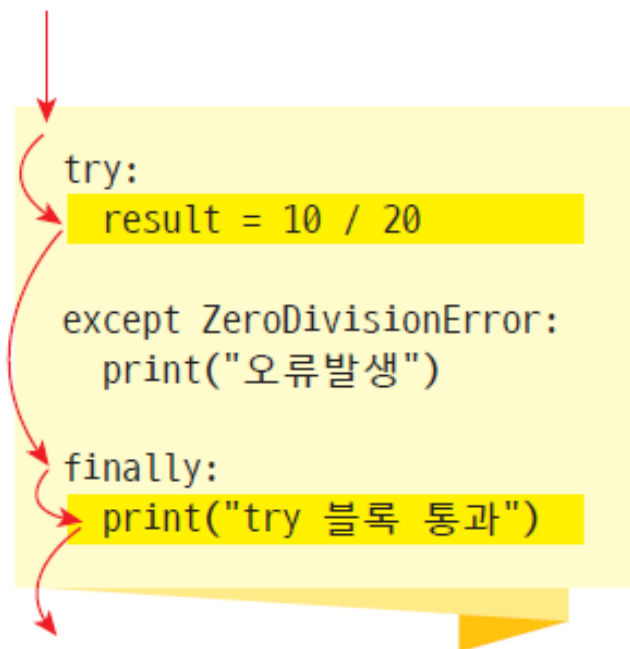
<exception04.py>

```
try:
    a = [1,2]
    print(a[3])
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except IndexError:
    print("인덱싱 할 수 없습니다.")
```

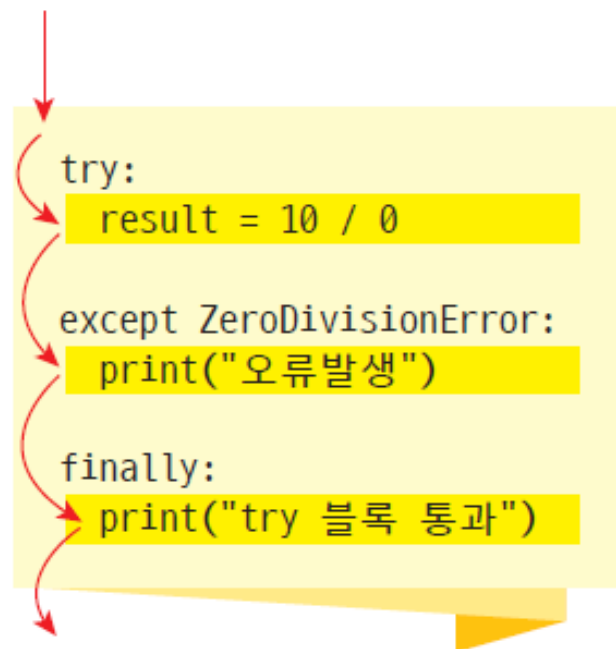
인덱싱 할 수 없습니다.

# finally 블록

- finally 블록 실행
  - 예외 발생 여부와 관계없이 항상 수행
  - 반드시 수행되어야 하는 문장을 추가



예외가 발생하지 않은 경우



예외가 발생한 경우



# try, except, finally

<finally01.py>

```
def calc(values):  
    sum = None  
    try:  
        sum = values[0] + values[1] + values[2]  
    except IndexError as err:  
        print("인덱스 에러: ", err)  
    except Exception as err:  
        print(err)  
    else:  
        print("에러 없음:", values)  
    finally:  
        print(f"sum={sum}")  
  
calc([1, 2, 3])  
calc([1, 2])
```

```
에러 없음: [1, 2, 3]  
sum=6  
인덱스 에러: list index out of range  
sum=None
```

# 핵심 정리

- 파일은 텍스트 파일과 이진 파일로 나뉘어진다.
- 파일은 `open()` 이후에 입출력이 끝나면 반드시 `close()`
- 파일에서 데이터를 읽거나 쓰는 함수는 `read()`와 `write()` 함수이다.
- 텍스트 파일에서 한 줄을 읽으려면 `for` 루프를 사용한다.
- 예외 처리는 오류가 발생했을 때 프로그램을 우아하게 종료하는 방법으로, `try` 블록과 `except` 블록으로 이루어진다.



# Questions?