

## 8.7 파일 입출력

- 파이썬을 이용한 파일 입출력에 대해서 알아보기

```
print('Hello World!')
```

Hello World! 라는 내용을 화면에 출력함  
이 내용을 파일로 저장하고자 함

## 8.7.0 경로

### • 경로 `path`

- 컴퓨터의 저장 장치 내에 데이터를 식별할 수 있는 위치
- `os.path` 모듈은 경로 이름을 다루는 유용한 함수들을 제공함
- `pathlib` 모듈은 파일 시스템의 경로를 나타내는 클래스들을 제공함

코드 8-7-0 : 파일 경로

file\_path.py

```
import os.path as path
file_path = '/usr/lib32/libc.so'
print(path.basename(file_path))      # libc.so
print(path.dirname(file_path))       # /usr/lib32
print(path.isfile(file_path))        # True
import os
for f in os.listdir():
    print(f)                          # libanl.so
```

코드 8-7-0 : 파일 경로

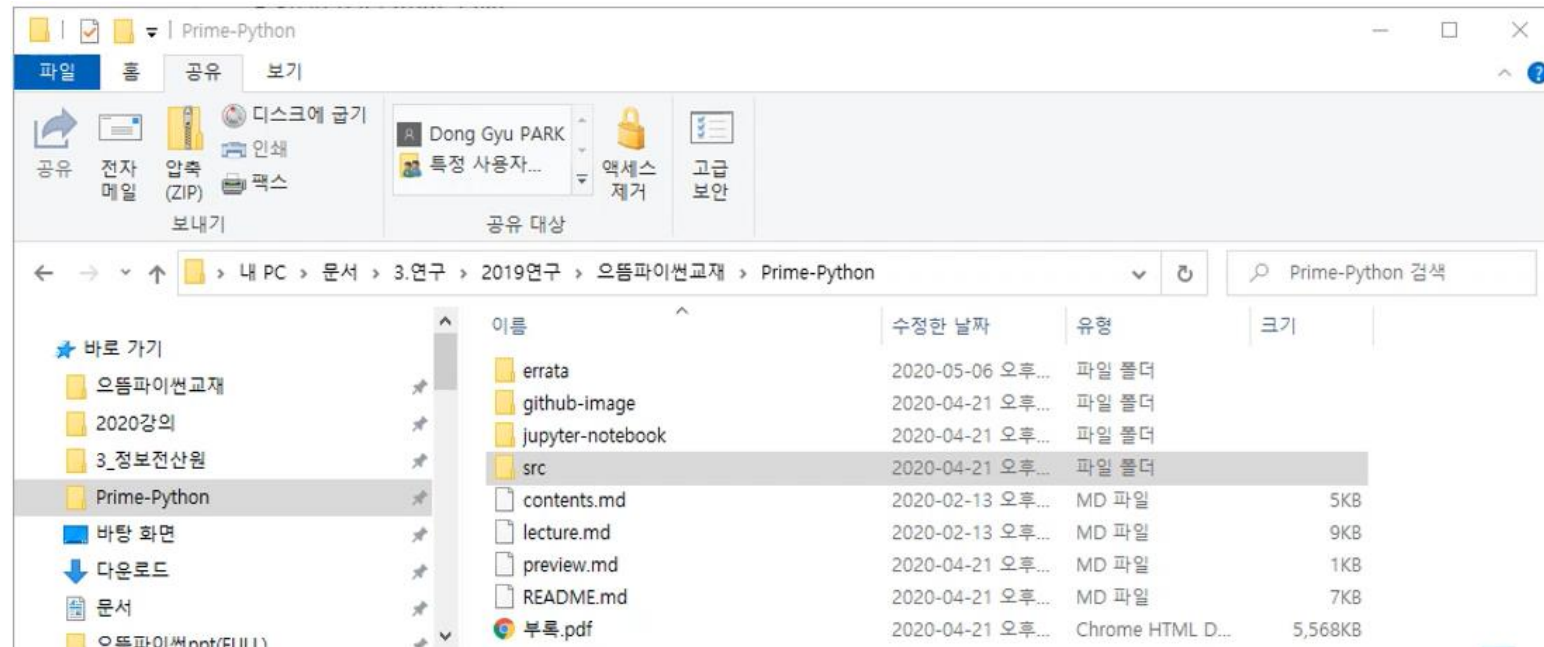
file\_path.py

```
from pathlib import Path
p = Path(file_path)
print(p.name)                        # libc.so
print(p.parent)                      # /usr/lib
print(p.is_file())                   # True
p = Path('/usr/lib32')
for f in p.iterdir():
    print(f)                          # /usr/lib32/libanl.a
```

## 8.7.1 파일이란 무엇인가

- 파일 **file**

- 컴퓨터의 저장 장치 내에 데이터를 저장하기 위해 사용하는 논리적인 단위
- 하드디스크 **hard disk**나 외장 디스크 **external disk**와 같은 저장장치에 저장한 후 필요할 때 다시 불러서 사용하는 것이 가능



# 파이썬에서 파일을 사용하기 위한 절차

## 1. 파일 열기

저장 장치 내의 위치와 파일 이름을 지정해서 파일을 가져온다.

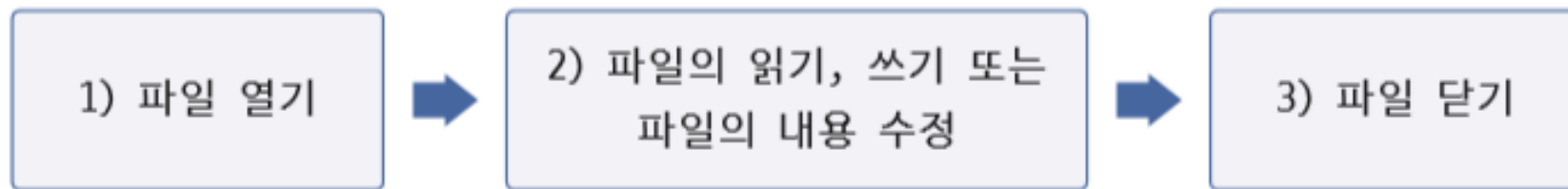
컴퓨터 저장장치 내의 파일 위치는 **경로** **path**라고 함

## 2. 사용목적에 따라 사용

내용 확인 및 새로운 내용 추가 또는 기존 내용 삭제 가능

## 3. 파일 닫기

파일을 모두 사용하고 나면 이 메모리를 시스템에서 사용할 수 있도록 반환



[그림 8-2] 파일을 열고 사용한 후 닫는 3단계의 절차

## 8.7.2 파일 쓰기과 모드

- open() 함수는 'hello.txt'라는 파일을 열어서 가져오는 역할
  - 'w'라는 파일 오픈 모드를 지정하여 가져옴 (쓰기 전용)
- open() 함수는 파일 객체를 반환
- 파일 객체 f는 write() 메소드를 이용하여 지정된 형식으로 파일에 지정된 텍스트를 쓰는 일을 한다.
- f.close() 메소드는 파일을 닫아줌

코드 8-10 : 파일 열기와 쓰기, 파일 닫기의 표현

file\_write\_hello.py

```
f = open('hello.txt', 'w')           # 1) 파일을 연다.  
f.write('Hello World!')              # 2) hello.txt 파일에 문자를 쓴다.  
f.close()                           # 3) 파일을 닫는다.
```

# open() 함수

- 파이썬에서 파일을 다루는 핵심 함수
- 파일 이름과 파일 오픈 모드의 인자를 가질 수 있음
- encoding이라는 인자를 통해 파일의 인코딩 형식 지정 가능
- 지정된 경로가 없을 경우 디폴트 경로는 파이썬 스크립트 파일이 있는 곳이 된다

모드	의미
r	<p>Read의 약자로 파일을 <b>읽기 전용 모드</b>로 연다. 이 모드는 파일 열기의 기본 모드이다.</p> <ul style="list-style-type: none"> <li>- 파일이 없을 경우 에러가 발생한다.</li> <li>- 읽기 전용 모드이므로 파일을 쓰려고 하면 오류가 발생한다.</li> </ul>
w	<p>Write의 약자로 파일을 <b>쓰기 전용 모드</b>로 연다.</p> <ul style="list-style-type: none"> <li>- 파일이 없을 경우 파일을 생성하며, 파일이 있을 경우 <b>덮어쓰게 된다(주의)</b>.</li> </ul>
a	<p>Append의 약자로 파일을 쓰기 모드로 열어서 기존 파일의 뒤에 새로 작성된 내용을 추가한다.</p> <ul style="list-style-type: none"> <li>- 기존 파일이 없을 경우 새로 파일을 만들어서 추가한다.</li> </ul>
x	<p>이 모드는 쓰기 전용으로 파일을 여는데, 파일을 새로 만들기 위해서 사용한다.</p> <ul style="list-style-type: none"> <li>- 이 모드는 a나 w와 달리 파일이 있을 경우 에러가 발생한다.(w 모드의 안전모드)</li> </ul>
+	<p>+ 기호는 읽기 쓰기 모드로 'r+'나 'w+', 'r+t'와 같은 조합으로 사용할 수 있다. 'r+'와 같이 표기하면 읽기 모드로 열어서 쓰기까지 가능하다. 반면 'w+'와 같이 표기하면 쓰기 모드로 열어서 읽기까지 가능하다.</p>

# 이진 파일을 다룰지 텍스트 파일을 다룰지를 지정하는 모드

파일 모드	의미
t	text의 약자로 텍스트 파일형식으로 파일을 열거나 생성한다. 이 모드는 파일 열기의 기본 모드이다.
b	binary의 약자로 이진 파일형식으로 파일을 열거나 생성한다.



```
f = open('hello.txt', 'w') # 파일 열기
```

파일을 쓰기모드(w)로 읽어서 파일 객체를 반환 : write() 메소드 호출하여 지정된 파일에 텍스트를 쓸 수 있음

```
f = open('hello.txt', 'wt') # 파일을 텍스트로 열기
```

위의 코드와 동일함 텍스트(t) 모드로 파일을 열어서 쓸 수 있음

```
f = open('hello.txt', 'wb') # 파일을 바이너리로 열기
```

wb모드는 바이너리 모드로 파일 쓰기 가능

```
f = open('hello.txt', 'r+t') # 텍스트 읽기 쓰기 모드
```

기본모드는 읽기 모드이지만 쓰기도 가능, 파일 없으면 오류 발생

```
f = open('hello.txt', 'w+t') # 텍스트 쓰기 읽기 모드
```

기본모드는 쓰기 모드이지만 읽기도 가능함

```
f = open('hello.txt', 'a+t') # 텍스트 추가
```

파일의 모든 내용을 남겨두고 맨 뒤에 추가함, 읽기도 가능하며 파일이 없을 경우 만든다

```
f.write('Hello World!') # Hello World!를 파일에 쓰기
```

```
f.write('Love of my life, you\'ve hurt me\n')
```

```
f.write('You\'ve broken my heart and now you leave me\n')
```

```
f.write('Love of my life, can\'t you see?\n')
```

\n을 입력하여야 파일 내 줄바꿈이 가능함.  
'..' 내부에 따옴표를 출력하려면 \'를 사용

```
print('Love of my life, you\'ve hurt me', file=f)
```

print 함수로도 가능함

- 컴퓨터의 입출력을 담당하는 프로세서에서 데이터는 **버퍼링 buffering**된 후 처리되는 것이 일반적
- 이때 사용되는 **버퍼 buffer**는 임시로 사용되는 기억공간
  - 파일을 효율적으로 쓰기 위해서 일정한 크기만큼의 데이터를 모아 두었다가 한꺼번에 읽거나 쓰는 용도
- `close()` 함수를 통해서 버퍼의 내용을 디스크로 보내고 버퍼 크기만큼의 메모리는 비워지게 된다.

```
f.close() # 파일을 닫는다.
```



#### NOTE : 버퍼의 필요성과 close() 함수

컴퓨터의 하드디스크를 통해 데이터를 읽고 쓰는 동작은 중앙처리장치나 메모리 내에서 데이터를 읽고 쓰는 것보다 수십 배에서 수백 배 이상 느리다. 따라서 가급적 데이터를 디스크에 읽고 쓰는 일을 한꺼번에 모아서 처리하는 것이 더 빠르다. 이 때문에 컴퓨터의 메모리에 있는 1000 바이트 데이터를 1 바이트씩 1000번 디스크에 저장하는 것 보다 100 바이트 단위로 10번 저장하는 것이 더 빠르다.

따라서 버퍼를 사용하여 write 명령으로 들어온 데이터를 100 바이트 단위로 모아 두었다가 100 바이트가 되면 한 번에 쓰는 것이 일반적으로 더 효과적이다. close() 함수가 하는 일은 이제 이 파일에 더 이상 write 명령을 보낼 일이 없다는 정보를 줌으로서 버퍼에 있는 정보를 디스크에 저장하는 역할을 한다.

## 8.7.3 파일 읽기

- read() 메소드를 통해서 파일 읽어오기
- 파일의 모든 문자열(전체 내용)은 줄 바꿈에 관계없이 하나의 문자열로 반환됨

코드 8-11 : 파일 열기와 읽기, 파일 닫기의 표현

hello.txt 파일을 읽어서 print() 함수로 출력하는 프로그램

file\_read\_hello.py

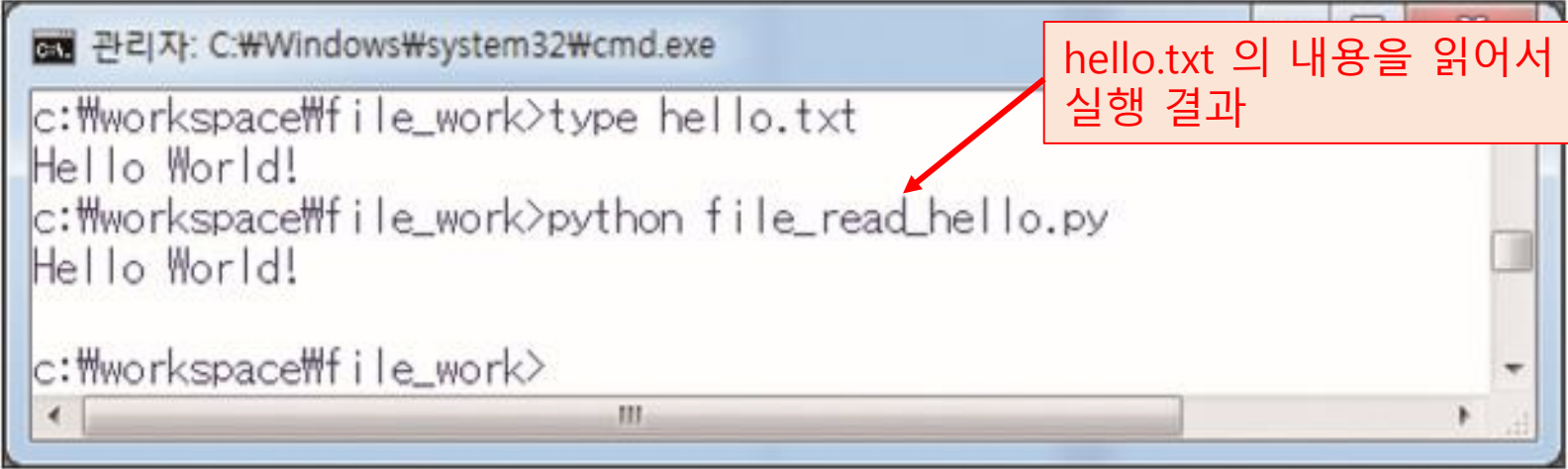
```
f = open('hello.txt', 'r')  
s = f.read()  
print(s)  
f.close()
```

# 파일을 연다.  
# hello.txt 파일을 읽는다.  
# 파일의 내용을 출력한다.  
# 파일을 닫는다.

실행결과

Hello World!

- file\_read\_hello.py 파일을 인터프리터에서 수행한 결과



The screenshot shows a Windows command prompt window with the title bar '관리자: C:\Windows\system32\cmd.exe'. The command history is as follows:  
c:\workspace\file\_work>type hello.txt  
Hello World!  
c:\workspace\file\_work>python file\_read\_hello.py  
Hello World!  
c:\workspace\file\_work>  
A red arrow points from a text box to the output 'Hello World!' of the second command.

hello.txt 의 내용을 읽어서 출력하는 프로그램의 실행 결과

**[그림 8-5]** file\_read\_hello.py 파일을 인터프리터에서 수행한 결과

- f.read() 메소드의 인자로 5를 주면 파일로부터 5개의 문자를 읽어 와서 이 문자를 문자열로 반환
- 'Hello'의 다섯 문자만 출력된다.

코드 8-12 : read() 메소드를 이용하여 지정된 문자 크기만큼 읽기

file\_read\_5char.py

```
f = open('hello.txt', 'r')           # 파일을 연다.  
s = f.read(5)                       # hello.txt 파일을 읽는다.  
print(s)                            # 파일의 내용을 출력한다.  
f.close()                           # 파일을 닫는다.
```

실행결과

Hello


- readline() 메소드를 이용하여 foo.txt 파일의 두 줄 읽은 후 출력

코드 8-13 : readline() 메소드를 이용한 줄 단위 읽기와 출력하기

file\_readline.py

```
f = open('foo.txt', 'r')      # 파일 열기
s = f.readline()              # 파일의 첫 번째 줄을 읽어온다
print(s, end='')              # 이 줄을 출력한다
s = f.readline()              # 파일의 두 번째 줄을 읽어온다
print(s, end='')              # 이 줄을 출력한다
f.close()                     # 파일을 닫는다
```

f.readline() 이 두번 나오므로 foo.txt 파일에서  
두 줄을 읽는다



foo.txt 파일의 내용

AAA

BBB

CCC

실행결과

AAA

BBB

- readlines() 메소드를 이용하여 foo.txt 파일 내용 읽기

코드 8-13 : readlines() 메소드를 이용한 읽기

file\_readline.py

```
f = open('foo.txt', 'r') # 파일 열기
for l in f.readlines():
    print(l.rstrip())
for l in f:
    print(l.rstrip())
f.close() # 파일을 닫는다
```

Read and return a list of lines from the stream.

foo.txt 파일의 내용

AAA

BBB

CCC

실행결과

AAA

BBB

CCC



- 스트링의 `rstrip()` 메소드를 사용하여 마지막에 나오는 문자 삭제 가능

코드 8-14 : `readline()` 메소드와 `rstrip()`을 이용한 줄 단위 읽기와 출력

file\_readline\_rstrip.py

```
f = open('foo.txt', 'r')      # 'foo.txt' 파일을 연다.
s = f.readline().rstrip()     # 'AAA' 줄을 읽고 오른쪽 줄 바꿈 문자를 지움
print(s)                     # 이 줄을 출력한다
s = f.readline().rstrip()     # 'BBB' 줄을 읽고 오른쪽 줄 바꿈 문자를 지움
print(s)                     # 이 줄을 출력한다
f.close()                    # 파일을 닫는다
```

## 8.7.4 파일 추가하기

- 'a+' 모드를 사용
  - 이미 만들어진 파일의 뒤쪽에 새로운 내용을 추가
- a는 추가(append)모드
- +는 기존 파일이 없을 경우 새롭게 파일을 생성하는 모드

## 코드 8-15 : 파일 추가(append) 모드를 이용한 파일 열기와 추가하기

file\_append.py

```
f = open('foo.txt', 'a+')    # 파일 열기
f.write('This will be appended.\n')
f.write('This too.\n')
f.close()
```

foo.txt 파일 : 파일 추가 전

AAA

BBB

CCC

foo.txt 파일 : 파일 추가 후

AAA

BBB

CCC

This will be appended.

This too.

## 8.7.5 파일 쓰기과 읽기 예제

- 실제로 사용가능한 응용분야를 찾아봅시다
- 사용자로 부터 다섯 개의 정수를 입력으로 받아서 이 정수를 data5.txt에 저장
- data5.txt를 읽어서 이 정수들의 합과 평균을 출력하기

코드 8-16 : 사용자로부터 입력받은 다섯 개의 정수를 저장하는 프로그램

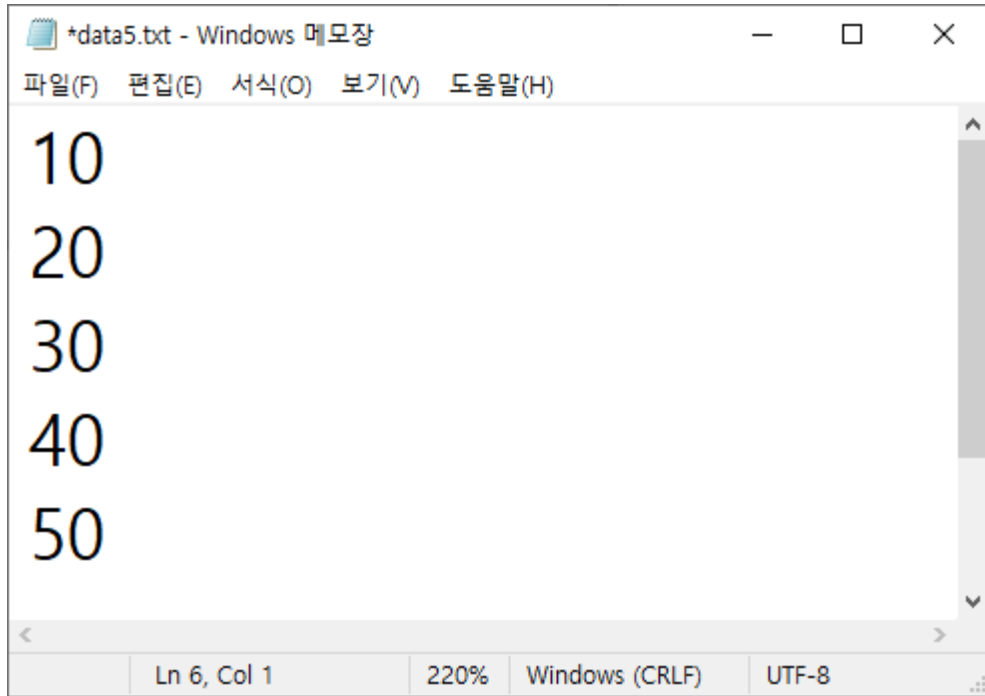
file\_write\_5num.py

```
f = open('data5.txt','w')           # 파일을 쓰기 모드로 연다.
for _ in range(5):
    n = input('정수를 입력하세요: ')
    f.write(n)                       # 입력된 값을 문자열로 쓰기
    f.write('\n')                   # 값을 쓴 후 줄바꿈하기
f.close()                           # 파일을 닫는다.
```

실행결과

```
정수를 입력하세요: 10
정수를 입력하세요: 20
정수를 입력하세요: 30
정수를 입력하세요: 40
정수를 입력하세요: 50
```

- 10, 20, 30, 40, 50을 입력하고 나서 file\_write5.py 파일이 있는 폴더를 살펴보면 다음과 같이 data5.txt 파일이 생성되어있음



- 파일로부터 정수를 읽어서 이 정수들의 합과 평균을 출력
- 파일에 저장된 10은 문자열 '10'이므로 int()를 이용하여 정수로 바꾸어야 덧셈 연산이 가능

코드 8-17 : 파일로 부터 정수를 입력받아 합과 평균을 계산

file\_read\_5num.py

```
f = open('data5.txt','r')                # 읽기 모드로 파일 열기
su = 0
for _ in range(5):
    n = int(f.readline())                # data5.txt 파일의 한 줄 내의 숫자를 읽어서
    su += n                             # su에 누적해서 더한다.

print('다섯 숫자의 합 = {}, 평균 = {}'.format(su, su/5)) # 합과 평균을 출력
f.close()                                           # 파일을 닫는다.
```

실행결과

다섯 숫자의 합 = 150, 평균 = 30.0

- 사용자로부터 파일 이름을 받아들이고 후 파일의 내용을 라인 번호와 함께 출력하는 기능을 파일 입출력을 통해서 구현하기

we\_will\_rock.txt 파일

Buddy, you're a boy, make a big noise

Playing in the street, gonna be a big man someday

You got mud on your face, you big disgrace

Kicking your can all over the place, singin'

We will, we will rock you

We will, we will rock you



## 코드 8-18 : 사용자로부터 입력 받은 파일을 번호를 붙여서 출력하기

file\_numbering1.py

```
fname = input('입력할 파일의 이름 : ')
f = open(fname, 'r')                # 파일 열기
n = 1                               # 줄 번호를 출력하기 위한 변수
l = f.readline()
while l :                           # 읽어들이 줄이 있으면 반복 수행함
    print('{:3d}: {}'.format(n, l), end = '') # 줄 번호와 내용을 출력
    n += 1                           # 줄 번호를 증가시킴
    l = f.readline()                # 다음 줄을 읽어온다

f.close()
```

### 실행결과

입력할 파일의 이름 : we\_will\_rock.txt

```
1: Buddy, you're a boy, make a big noise
2: Playing in the street, gonna be a big man someday
3: You got mud on your face, you big disgrace
4: Kicking your can all over the place, singin'
5: We will, we will rock you
6: We will, we will rock you
```

영문으로된 텍스트 파일을 읽어들이는  
데 문제가 없다

- format() 메소드를 사용하여 줄 번호와 줄의 내용을 출력
- 한글이 포함된 파일을 제대로 처리할 수 없다

- 프로그램의 입력으로 앞서 작성한 'file\_read\_5num.py'를 사용
- UnicodeDecodeError라는 에러가 발생
  - 파일의 내부에 한글이 포함되어 있어 이 한글 코드를 디코딩 할 수 없을 때 발생하는 오류

#### 실행결과

입력할 파일의 이름 : file\_read\_5num.py

Traceback (most recent call last):

File "C:\workspace\file\_work\file\_numbering1.py", line 6, in <module>

```
l = f.readline()
```

UnicodeDecodeError: 'cp949' codec can't decode byte 0xed in position 29: illegal multibyte sequence

한글이 포함된 파일의 경우 cp949 형식의 코덱  
만으로는 한글을 디코딩 할 수 없음

file\_read\_5num.py

```
f = open('data5.txt', 'r')
```

```
su = 0
```

```
for _ in range(5):
```

# 읽기 모드로 파일 열기

- open() 함수의 인코딩 방식을 encoding = 'UTF8'과 같이 키워드로 지정하여 문제를 해결

한글이 포함된 텍스트 파일을 읽어들이는데 문제가 없다

```
f = open(fname, 'r', encoding = 'UTF8') # UTF8 인코딩을 통해 파일 열기
```

실행 결과 : open() 함수에서 encoding = 'UTF8' 인코딩을 지정한 결과

입력할 파일의 이름 : file\_read\_5num.py

```
1: f = open('data5.txt','r') # 파일 열기
2: su = 0
3: for _ in range(5):
4:     n = int(f.readline()) # data5.txt 파일의 숫자를 읽어서
5:     su += n # su에 누적해서 더한다
6:
7: print('다섯 숫자의 합 = {}, 평균 = {}'.format(su, su/5) # 합과 평균을 출력
8: f.close()
```

- 파일 이름이 잘못 입력될 경우 발생하는 문제
  - file\_read\_5num과 같이 확장자를 생략하고 입력하면 open() 함수는 'FileNotFoundError' 오류를 출력

#### 실행결과

입력할 파일의 이름 : file\_read\_5num

입력할 파일의 이름 : file\_read\_5num

Traceback (most recent call last):

File "C:/workspace/file\_work/file\_numbering3.py", line 4, in <module>

f = open(fname, 'r', encoding='UTF8') # 파일 열기

FileNotFoundError: [Errno 2] No such file or directory: 'file\_read\_5num'

안전한 코딩은 중요하다!  
따라서 예외처리가 필요하다!!

## 코드 8-19 : try-except 문을 사용한 파일 입출력

file\_numbering3.py

```
import sys
fname = input('입력할 파일의 이름 : ')
try :
    f = open(fname, 'r', encoding='UTF8')
except IOError:
    print('Could not read file:', fname)
    sys.exit()
except:
    print('Unexpected error:', sys.exc_info()[0])
    sys.exit()

n = 1
l = f.readline()
while l :
    print('{:3d}: {}'.format(n, l), end='')
    n += 1
    l = f.readline()
f.close()
```

파이썬 프로그램을 종료함

발생한 예외를 반환함

# 파일 열기

# n을 1로 초기화 함

# 변수 l은 읽어들이 한 줄의 문자열을 저장합니다

# 한 줄을 출력한 후 줄 수를 증가시킨다

# 다음 줄을 읽어온다

- try-except 문을 이용하여 파일명이 제대로 입력되었는지, 혹은 파일을 열 수 있는가를 검사한 후 오류가 없을 경우 파일을 읽어서 출력해야 함

### 실행결과

입력할 파일의 이름 : file\_read\_5num

Could not read file: file\_read\_5num

## 8.8 with 문법

- 문법을 명확하게 만들고 예외를 손쉽게 처리하는 방법
- **컨텍스트 매니저** `context manager`에 의해서 실행되는 `__enter__()`과 `__exit__()` 함수를 정의하여, `with` 구문 몸체의 앞부분과 뒷부분에 실행되는 코드를 대신해줌
- `with` 구문 이용 시 `try-except-finally`를 대신하여 더욱 쉽게 사용 가능

코드 8-20 : 파일 열기와 쓰기, 파일 닫기의 표현 1

```
file_write_hello1.py
```

```
f = open('hello.txt', 'w') # 파일 열기
f.write('Hello World!') # hello.txt 파일에 쓰기
f.close() # 파일을 닫는다.
```

- f.write() 작업은 항상 오류의 가능성을 포함하고 있기 때문에 try 절을 이용하여 다음과 같이 수정

코드 8-21 : 파일 열기와 쓰기, 파일 닫기의 표현 2

file\_write\_hello2.py

```
f = open('hello.txt', 'w')                # 파일 열기
try:
    f.write('Hello World!')                # hello.txt 파일에 쓰기
finally:
    f.close()                             # 파일을 닫는다.
```

- with ~ as : 문을 사용하여 간단하게 사용

코드 8-22 : 파일 열기와 쓰기, 파일 닫기의 표현 3

file\_write\_hello3.py

```
with open('hello.txt', 'w') as f:           # 파일 열기와 닫기를 자동 수행함
    f.write('Hello World!')                 # hello.txt 파일에 쓰기
```

- 파일 쓰기가 완료되면 자동으로 f.close()가 수행되기 때문에 finally절을 사용할 필요가 없음



## 코드 8-23 : try-except 문을 사용한 파일 입출력

file\_numbering3.py

```
import sys
success = True
fname = input('입력 파일명 : ')
try :
    f = open(fname, 'r', encoding='UTF8')    # 파일 열기를 위한 open()
except IOError:
    print('Could not read file:', fname)
    success = False
except:
    print('Unexpected error:', sys.exc_info()[0])
    success = False
if success :
    n = 1
    l = f.readline()
    while l :
        print('{:3d}: {}'.format(n, l), end='')
        n += 1
        l = f.readline()
    f.close()    # open() 함수와 너무 멀리 떨어져있음
print('file access successful? ', success)
```

- 옆의 코드를 open() 구문으로 간결하게 수정하기
- 파일 읽기에 성공시 f.readline()을 통해서 파일을 라인단위로 읽어서 매 라인마다 라인 번호를 출력하는 while문이 수행됨


## 코드 8-24 : try-except 문을 사용한 파일 입출력

file\_numbering4.py

```
import sys
success = False
try:
    fname = input('입력 파일명: ')
    with open(fname, 'r', encoding = 'UTF8') as f:
        n = 1
        l = f.readline()
        while l:
            print('{:3d}: {}'.format(n, l), end='')
            n += 1
            l = f.readline()
        success = True
except IOError:
    print('Could not read file:', fname)
except:
    print('Unexpected error:', sys.exc_info()[0])

print('file access successful? ', success)
```

파이썬 컨텍스트 매니저가 파일 객체를 반환하며, 이 절이 끝나면 자동으로 파일의 close() 메소드를 호출함





Questions?