# 09. 파일 I/O

## ❖ 파일 액세스

- 자원 누수 방지를 돕는 with ~ as
- with문의 비밀 : 컨텍스트 매니저
- open() 함수 다시 보기

## ❖ 텍스트 파일 읽기/쓰기

- 문자열을 담은 리스트를 파일에 쓰는 writelines() 메소드
- 줄 단위로 텍스트를 읽는 readline()과 readlines() 메소드
- 문자 집합과 인코딩에 대하여

## ❖ 바이너리 파일 다루기

# 파일 액세스

- 어플리케이션이 운영체제에게 파일 처리를 API 함수를 통해 의뢰하면, 운영체제가 요청한 업무를 수행해주고 그 결과를 어플리케이션에게 돌려줌.
- 어플리케이션이 파일 처리 업무를 의뢰하는 과정과 각 과정에서 사용되는 파이썬 함수

# 파일 액세스

## ❖ 예제 : 11/write.py

- 열기-쓰기-닫기의 순서로 함수 호출 수행
- write.py를 실행하고 나면 test.txt 파일을 열어 내용 확인

```
file = open('test.txt', 'w')
file.write('hello')
file.close()
```

```
f = open('test.txt', 'w', encoding='utf-8')
for i in range(1, 10):
    f.write("%d: Life is too short, You need Python₩n" % i)
f.close()
```

## ❖ 예제 : 11/read.py

```
file = open('test.txt', 'r')
str = file.read()
print(str)
file.close()
```

**read 와 write 는 모든 파일에 사용**
**read( ) - 전부 읽어 옴**
**read(n) - n 바이트 읽어 옴**

# 파일 액세스 – 자원 누수 방지를 돕는 with ~ as

- open() 함수와 함께 with ~ as문을 사용하면 명시적으로 close() 함수를 호출하지 않아도 파일이 항상 닫힘.
- with ~ as 문을 사용하는 방법은 다음과 같음.

```
with open(파일이름) as 파일객체:
    # 코드블록
    # 이곳에서 읽거나
    # 쓰기를 한 후
    # 그냥 코드을 빠져 나가면 됩니다.
```

*"파일객체 = open(파일이름)"* 와 같다고 생각하면 됩니다.

with 문 덕분에 close()를 하지 않아도 됩니다.

- 예제 : 11/openwith.py

```
with open('test.txt', 'r') as file:
    str = file.read()
    print(str)
    #file.close()
```

# 열라, 읽으라(쓰라), 그리고 닫으라 – open() 함수 다시 보기

## ❖ open() 함수의 매개 변수는 모두 8개

- 하나의 필수 매개변수와 일곱 개의 매개변수
- 반환값은 물론 파일 객체.

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
newline=None, closefd=True, opener=None)
```

## ❖ file

- 파일의 경로를 나타내는 문자열 또는 이미 생성해놓은 파일 객체의 파일 기술자(file 객체의 fileno)

## ❖ mode (파일 열기 모드)

| 문자 | 의미 |
|---|---|
| 'r' | 읽기용으로 열기 (기본값) |
| 'w' | 쓰기용으로 열기. 이미 같은 경로에 파일이 존재하면 파일내용을 비움. |
| 'x' | 배타적 생성모드로 열기. 파일이 존재하면 IOError 예외 일으킴. |
| 'a' | 쓰기용으로 열기. 단, 'w'와는 달리 이미 같은 경로에 파일이 존재하는 경우 기존 내용에 덧붙이기를 함. |
| 'b' | 바이너리 모드 - 바이트 단위 |
| 't' | 텍스트 모드(기본값) - 문자열 단위 |
| '+' | 읽기/쓰기용으로 파일 읽기 |

# 열라, 읽으라(쓰라), 그리고 닫으라 – open() 함수 다시 보기

## buffering

- 0을 입력하면 파일 입출력시에 버퍼링을 수행하지 않으며(바이너리 모드에서만 사용 가능), 1을 입력하면 개행 문자(₩n)를 만날 때까지 버퍼링을 하는 라인(line) 버퍼링을 수행(텍스트 모드에서만 사용 가능).
- 임의의 값으로 직접 버퍼의 크기를 지정하고 싶을 때는 이 매개변수에 1보다 큰 수를 입력.

## encoding

- 텍스트 모드에서만 사용. <문자 집합과 인코딩>에서 별도 설명

## erros

- 텍스트 모드에서만 사용. 인코딩/디코딩 수행시의 에러 처리 옵션임.

| errors | 의미 |
|---|---|
| 'strict' | 인코딩 에러가 발생할 때 ValueError 예외를 일으킵니다. None과 똑같은 효과를 냅니다. |
| 'ignore' | 이름 그대로 에러를 무시합니다. |
| 'replace' | 기형적인 데이터가 있는 곳에 대체 기호(예를 들어 '?')를 삽입합니다. |
| 'surrogateescape' | U+DC80 ~ U+DCFF 사이에 있는 유니코드 사용자 자유 영역(PUA)의 잘못된 바이트를 코드 포인트(code points)로 나타냅니다. (코드 포인트는 문자를 나타내는 번호입니다. 나중에 한 번 더 설명하겠습니다.) |
| 'xmlcharrefreplace' | 파일에 기록하려는 텍스트 안에 지정된 인코딩(문자를 읽고 쓰는 방식을 정의한 규칙. 나중에 자세히 설명하겠습니다.)에서 지원되지 않는 문자를 &#NNN; 꼴의 XML 문자 참조로 바꿔서 기록하는 옵션입니다. 파일을 쓸 때만 적용됩니다. |
| 'backslashreplace' | 이 옵션도 역시 파일에 텍스트를 기록할 대만 사용됩니다. 현재 인코딩에서 지원되지 않는 문자를 역슬래쉬로 시작되는 이스케이프 시퀀스로 바꿔서 기록합니다. |

# 텍스트 파일 읽기/쓰기 - 문자열을 담은 리스트를 파일에 쓰는 write() 메소드

❖ **예제 : 11/writelist.py(for와 write() 메소드를 이용)**

```
lines = ["we'll find a way we always have - Interstellar₩n",
         "I'll find you and I'll kill you - Taken₩n",
         "I'll be back - Terminator 2₩n"]

with open('movie_quotes.txt', 'w') as file:
    for line in lines:
        file.write(line)
```

- 실행 결과

```
>writelist.py

>type movie_quotes.txt
we'll find a way we always have - Interstellar
I'll find you and I'll kill you - Taken
I'll be back - Terminator 2
```

type은 윈도우에서 제공하는 명령어입니다. 리눅스 또는 유닉스에서는 비슷한 기능의 명령어로 cat이 있습니다.

# 텍스트 파일 읽기/쓰기 - 문자열을 담은 리스트를 파일에 쓰는 writelines() 메소드

❖ **예제 : 11/writelines.py(writelines() 메소드를 이용)**

```
lines = ["we'll find a way we always have - Interstellar₩n",
         "I'll find you and I'll kill you - Taken₩n",
         "I'll be back - Terminator 2₩n"]

with open('movie_quotes.txt', 'w') as file:
    file.writelines(lines)
```

- 실행 결과

```
>writelines.py

>type movie_quotes.txt
we'll find a way we always have - Interstellar
I'll find you and I'll kill you - Taken
I'll be back - Terminator 2
```

# 텍스트 파일 읽기/쓰기 - 줄 단위로 텍스트를 읽는 readline()
# 한 행을 읽어 옴

## ❖ 예제 : 11/readline.py (readline() 메소드 이용)

```
with open('movie_quotes.txt', 'r') as file:
    line = file.readline()
    while line != '':
        print(line, end='')
        line = file.readline()
```

readline() 메소드는 파일의 끝에 도달하면 '' 를 반환합니다. 그런데 실제로 빈 줄을 읽어 들였을 때는 어떡하냐고요? 빈 줄을 읽어 들인 경우엔 개행문자를 반환합니다.

- 실행 결과

```
>readline.py
we'll find a way we always have - Interstellar
I'll find you and I'll kill you - Taken
I'll be back - Terminator 2
```

```
f = open('multilines.txt', 'r')
while True:
    line = f.readline()
    if not line:
        break # 무한루프 탈출
    print(line)
f.close()
```

# 텍스트 파일 읽기/쓰기 : 줄 단위로 텍스트를 읽는 readlines()
# 모든 행을 읽어 옴

## ❖ 예제 : 11/readlines.py (readlines() 메소드 이용)

```
with open('movie_quotes.txt', 'r') as file:
    lines = file.readlines()
    line = ''
    for line in lines:
        print(line, end='')
```

- 실행 결과

```
>readlines.py
we'll find a way we always have - Interstellar
I'll find you and I'll kill you - Taken
I'll be back - Terminator 2
```

## ❖ 기타

- print(문자열, 파일 객체, 구분자, 문자열 끝)
  - 파일 객체 : 기본값은 표준 출력 파일
    구분자 : 기본값은 공백( ‘ ’ ) → 각 인자 뒤에 공백 추가
    문자열 끝 : 기본값은 줄바꿈( ‘\n’ ) → 각 문자열 뒤에 줄바꿈 추가
    ex) fout = open( ‘test.txt’ , ‘wt’ )
    print(temp, file=fout, sep= ‘ ’ , end= ‘ ’ )
    cf) write는 공백이나 줄바꿈을 자동적으로 추가 하지 않음

# 텍스트 파일 읽기/쓰기 – 문자 집합과 인코딩에 대하여

- **“악보를 컴퓨터가 이해할 수 있는 기호로 표현할 수 있다면 컴퓨터가 음악 분야에서도 사용될 것” – Ada Lovelace**

- **문자 집합(Character Set) – 텍스트를 기호로 표현한 것.**
- **인코딩 – 문자열을 바이트로 변환**
- **디코딩 – 바이트를 문자열로 변환**

- **ASCII(American Standard Code for Information Interchange)**
  - 미국 정보 교환 표준 부호
  - 1960년대에 제정된 문자 집합으로, 이후에 개발된 문자 집합들의 토대가 됨.
  - ASCII는 7비트만을 이용하여 음이 아닌 수(0~127)에 문자 집합 내의 문자를 할당. 예) 61에는 ‘=’를, 65에는 ‘A’를, 97에는 ‘a’를 할당
  - 52개의 알파벳 대소문자(A~Z, a~Z), 10개의 숫자(0~9), 32개의 특수 문자(!@#$?... 등등), 하나의 공백 문자, 그리고 33개의 출력 불가능한 제어문자, 모두 128개의 문자를 표현.

# 텍스트 파일 읽기/쓰기 - 문자 집합과 인코딩에 대하여

## ❖ 유니코드(Unicode)

- 문자 집합 하나로 모든 문자를 표현할 수 있게 하는 것이 목적
- 초기에는 전세계의 언어별 문자들을 2바이트 안에서 영역을 나눠 할당
- 유니코드에서 문자에 부여되는 번호를 코드 포인트(Code Point)라고 함.
  - 코드포인트는 "U+" 뒤에 2바이트의 수를 16진수로 표현하여 붙여 표시
    예) 'A'의 코드 포인트는 U+0041, '한'의 코드 포인트는 U+D55C, '글'의 코드 포인트는 U+AE00
  - U+0000부터 U+007F까지를 ASCII와 동일하게 맞춰두고 그 뒷번호부터 각국의 문자를 할당.
- 누락된 현대 문자와 기호를 추가적으로 할당하고 고대 문자와 음악 기호 등을 추가하자 코드포인트가 2바이트를 넘어서게 됨.

## ❖ UTF(Unicode Transformation Format)

- 유니코드 변환 인코딩 형식
- UTF-8은 코드포인트의 크기에 따라 1바이트에서부터 4바이트까지 가변폭으로 인코딩하므로 1 바이트로 표현 가능한 U+0000(십진수 0)부터 U+007F(십진수 127)까지는 ASCII와 완벽하게 호환
  - UTF-8 인코딩 방식으로 저장된 문서는 유니코드를 알지 못하는 시스템에서도 사용 가능.
- UTF-8 외에도 UTF-7, UTF-16, UTF-32 인코딩 등이 있음.
- 참고 ; 1바이트-ASCII, 2바이트-라틴어, 3바이트-기본 다국어외 나머지, 4바이트-아시아언어+기호외 나머지

# 텍스트 파일 읽기/쓰기 – 문자 집합과 인코딩에 대하여

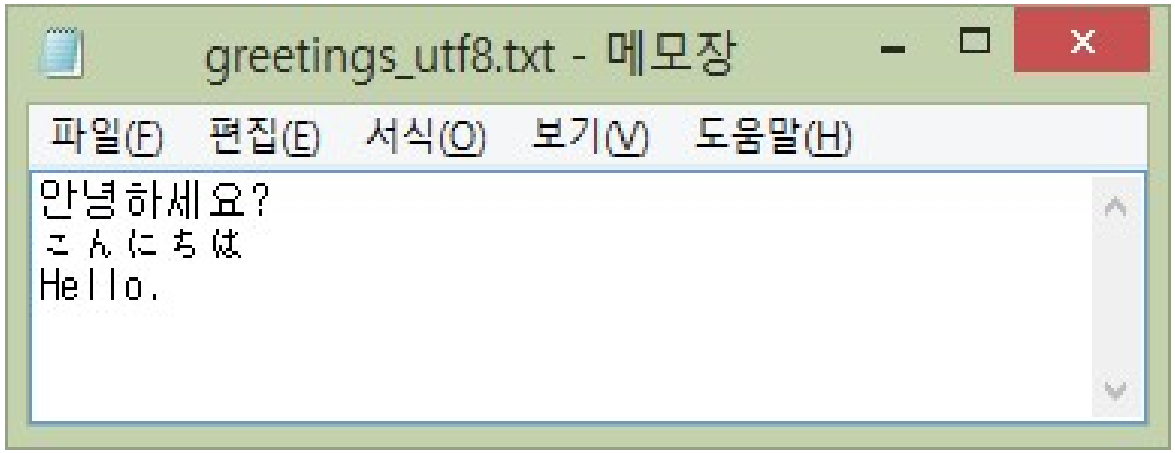
## ❖ 예제 : 11/utf8_write.py

```
lines = ['안녕하세요?₩n',
         'こんにちは₩n',
         'Hello.₩n']

with open('greetings_utf8.txt', 'w', encoding='utf-8') as file:
    for line in lines:
        file.write(line)
```

- 실행 결과

```
>utf8_write.py
```



# 텍스트 파일 읽기/쓰기 - 문자 집합과 인코딩에 대하여

## ❖ 예제 : 11/utf_read.py (UTF-8로 텍스트 파일 읽기)

```
with open('greetings_utf8.txt', 'r', encoding='utf-8') as file:
    lines = file.readlines()
    line = ''
    for line in lines:
        print(line, end='')
```

- 실행 결과

```
>utf8_read.py
안녕하세요?
こんにちは
Hello
```

## ❖ 예제 : 11/ascii_read.py (ASCII로 텍스트 파일 읽기)

```
with open('greetings_utf8.txt', 'r', encoding='ascii', errors='ignore') as file:
    lines = file.readlines()
    line = ''
    for line in lines:
        print(line, end='')
```

errors 옵션에 'ignore'를 지정해서 해당 인코딩으로 처리가 안되는 문자열이 있을 때는 지나치도록 했습니다.

- 실행 결과

```
>ascii_read.py
?

Hello.
```

errors 옵션에 'ignore'를 지정해서 해당 인코딩으로 처리가 안되는 문자열이 있을 때는 무시하도록 했습니다. 그 결과 ASCII와 호환 가능한 'Hello.' 말고는 아무 것도 출력되지 않습니다.

# 바이너리 파일 다루기

## ❖ struct 모듈

- 일반 데이터 형식과 bytes(이진 바이트열)형식 사이의 변환을 수행하는 함수 정의.
- “65” → “65” (텍스트 파일)  65 → “A” (텍스트 파일)
- 65 → 65 (바이너리 파일 – struct 모듈 통해서 원본 그대로 저장 가능)

## ❖ 실습 1 (struct 모듈 사용 예)

```
>>> import struct
>>> packed = struct.pack('i', 123)
>>> for b in packed:
        print(b)


123
0
0
0
>>> unpacked = struct.unpack('i', packed)
>>> unpacked
(123,)
>>> type(unpacked)
<class 'tuple'>
```

pack() 함수는 첫 번째 매개변수 ‘i’ 따라 4바이트 크기의 bytes 객체 packed를 준비하고 두 번째 매개변수를 packed에 복사해 넣습니다.

bytes 객체 packed의 각 바이트에 있는 내용을 출력합니다.

unpack() 함수는 튜플 형식을 반환합니다.

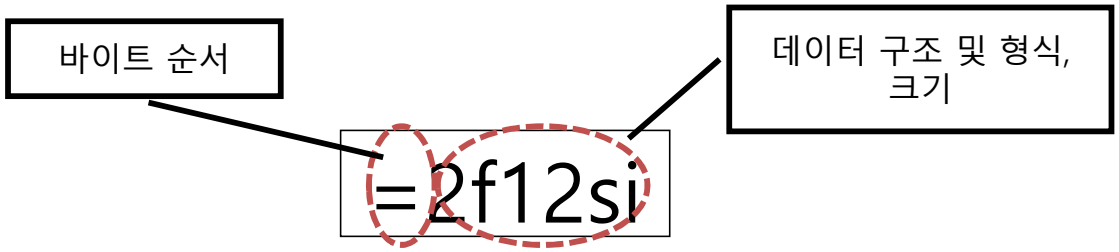
# 바이너리 파일 다루기

## ❖ pack() 함수와 unpack() 함수

struct.pack(fmt, v1, v2, ...) # 일반 데이터 ➔ bytes 형식 변환
struct.unpack(fmt, buffer) # bytes 형식 ➔ 튜플(데이터) 형식 변환

## ❖ pack() 함수와 unpack() 함수의 첫 번째 매개 변수 – fmt

- 데이터의 구조를 나타내는 형식 문자열(Format String)




## ❖ fmt에 바이트 순서를 지정하기 위해 사용할 수 있는 문자

| 문자 | 바이트 순서 | 크기 | 바이트 정렬 |
|---|---|---|---|
| @ | 시스템 바이트 순서를 따릅니다. | 시스템을 따릅니다. | 시스템을 따릅니다. |
| = | 시스템 바이트 순서를 따릅니다 | 표준을 따릅니다. | 수행 안 함. |
| < | 리틀 엔디안(오른쪽에서 왼쪽으로 채워짐) x86 계열 | 표준을 따릅니다. | 수행 안 함. |
| > | 빅 엔디안(왼쪽에서 오른쪽으로 채워짐) ARM 계열 | 표준을 따릅니다. | 수행 안 함. |
| ! | 네트워크 바이트 순서(빅 엔디안과 동일) | 표준을 따릅니다. | 수행 안 함. |

**c.f.) 0x12345678 → 78 56 34 12 (리틀 엔디안) 12 34 56 78 (빅 엔디안)**
**0x12 → 00 00 00 12 (리틀 엔디안) 12 00 00 00 (빅 엔디안)**
**c.f.) 바이트 정렬 : 자료형의 크기에 상관 없이 무조건 4바이트 또는 8바이트로 묶는 것 (다음 페이지 참조)**

# 바이너리 파일 다루기

## ❖ 데이터 구조를 정의하는 형식 문자

| 분류 | 문자 | 파이썬 자료형 | C 자료형 | 크기 |
|---|---|---|---|---|
| 정수형 | b | integer | signed char | 1 |
| | B | integer | unsigned char | 1 |
| | ? | bool | _Bool | 1 |
| | h | integer | short | 2 |
| | H | integer | unsigned short | 2 |
| | i | integer | int | 4 |
| | I | integer | unsigned int | 4 |
| | l | integer | long | 4 |
| | L | integer | unsigned long | 4 |
| | q | integer | long long | 8 |
| | Q | integer | unsigned long long | 8 |
| | n | integer | ssize_t | |
| | N | integer | size_t | |
| 부동 소수형 | f | float | float | 4 |
| | d | float | double | 8 |
| bytes | s | bytes | char[] | |
| | p | bytes | char[] | |
| | c | 크기가 1인 bytes형 | char | 1 |
| 기타 | x | no value | 패딩 바이트( 데이터 저장 용도가 아닌 바이트 배열을 맞추기 위한 자리 맞추기 용도) | |
| | P | integer | void * | |

# 바이너리 파일 다루기

## ❖ 실습 1 (부동 소수형 pack/unpack)

```
>>> import struct
>>> packed = struct.pack('f', 123.456)
>>> unpacked = struct.unpack('f', packed)
>>> unpacked
(123.45600128173828,)
```

## ❖ 실습 2 (문자열 pack/unpack)

```
>>> import struct
>>> packed = struct.pack('12s', '대한민국'.encode())
>>> unpacked = struct.unpack('12s', packed)
>>> unpacked[0].decode()  #튜플로 변환되어서 저장되기 때문에 unpacked[0] 디코딩
'대한민국'
```

**c.f) str.encode() / str.decode() 에서 함수 인자가 없으면 utf-8 변환**

## ❖ 실습 3 (구조체 pack/unpack)

```
>>> import struct
>>> packed = struct.pack('2d2i', *(123.456, 987.765, 123, 456))
>>> unpacked = struct.unpack('2d2i', packed)
>>> unpacked
(123.456, 987.765, 123, 456)
```

**c.f.) * : 튜플, 리스트의 요소를 하나씩 분리해서 매개 변수로 만들어 주는 연산자**

# 바이너리 파일 다루기

## ❖ 예제 : 11/binary_write.py

```
import struct

struct_fmt = '=16s2fi' # char[16], float[2], int
city_info = [
    # CITY,          Latitude,   Longitude,  Population
    ('서울'.encode(encoding='utf-8'), 37.566535, 126.977969, 9820000),
    ('뉴욕'.encode(encoding='utf-8'), 40.712784, -74.005941, 8400000),
    ('파리'.encode(encoding='utf-8'), 48.856614, 2.352222,   2210000),
    ('런던'.encode(encoding='utf-8'), 51.507351, -0.127758,  8300000)
    ]

with open('cities.dat', 'wb') as file:
    for city in city_info:
        file.write(struct.pack(struct_fmt, *city))
```

- 실행 결과

```
>binary_write.py
```

## ❖ 예제 : 11/binary_read.py

```
import struct

struct_fmt = '=16s2fi' # char[16], float[2], int
struct_len = struct.calcsize(struct_fmt)  #28-byte

cities = []
with open('cities.dat', "rb") as file:
    while True:
        buffer = file.read(struct_len)
        if not buffer: break #파일의 끝을 읽으면 빠져나옴
        city = struct.unpack(struct_fmt, buffer)
        cities.append(city)

for city in cities:
    name = city[0].decode(encoding='utf-8').replace('₩x00', '')
    print('City:{0}, Lat/Long:{1}/{2}, Population:{3}'.format(name,
    city[1], city[2], city[3]))
```

pack() 하는 과정에서 문자를 할당하고, 남은 공간에 채워진 ₩x00를 디코딩한 후 빈 문자열로 다시 바꿔 넣습니다

- 실행 결과

```
>binary_read.py
City:서울, Lat/Long:37.56653594970703/126.97796630859375,
Population:9820000
City:뉴욕, Lat/Long:40.71278381347656/-74.00594329833984,
Population:8400000
City:파리, Lat/Long:48.85661315917969/2.352221965789795,
Population:2210000
City:런던, Lat/Long:51.50735092163086/-0.1277579963207245,
Population:8300000
```