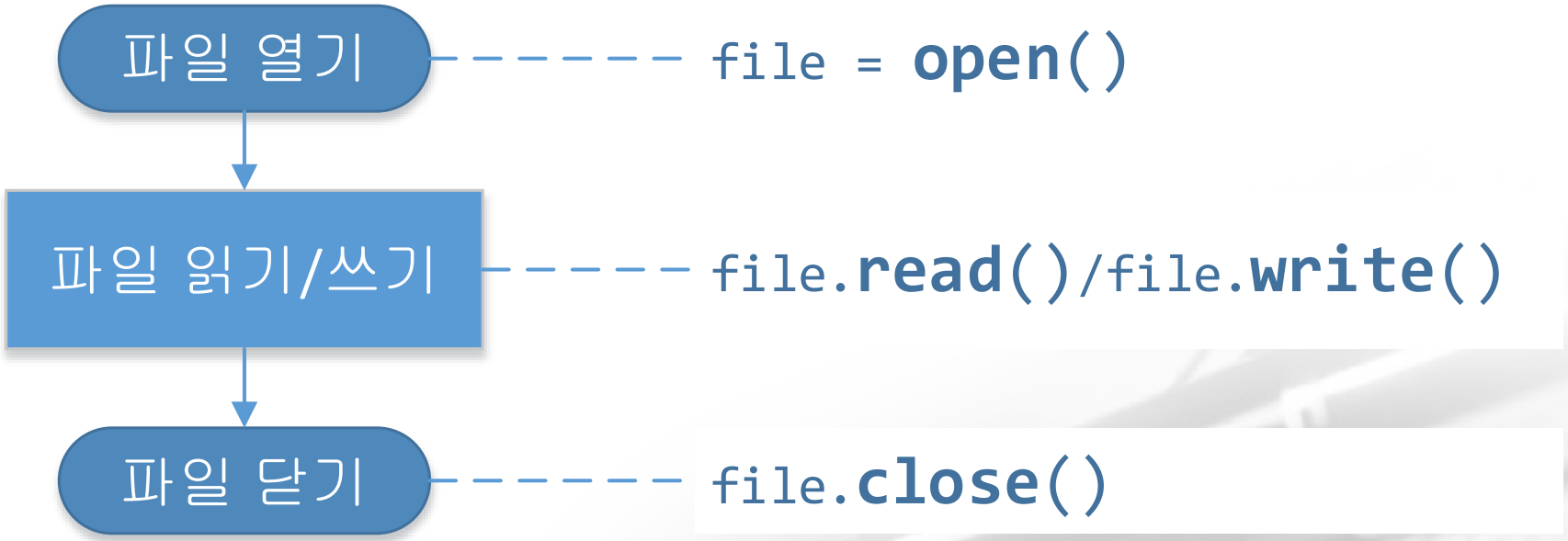
파일 처리

파일 처리

- 파일 처리
  - ✓ Application이 운영체제에게 파일 처리를 API 함수를 통해 요청하면 운영체제가 요청한 업무를 수행해주고 그 결과를 Application에게 돌려줌
  - ✓ Application이 파일 처리 업무를 의뢰하는 과정과 각 과정에서 사용되는 파이썬 함수



파일 처리

- open() 함수
  - ✓ 파일 쓰기
    - 파일에 기록하는 경우에는 2개나 3개의 매개변수를 대입
    - 첫번째 매개변수는 파일 경로이고 두번째는 'w' 이고 세번째는 인코딩 방식
    - 인코딩 방식의 경우 생략이 가능한데 생략을 하게 되면 시스템의 기본적인 문자 인코딩으로 기록
    - open()으로 연 파일 객체에 기록을 할 때는 write 함수를 이용해서 기록할 내용을 전달
    - ₩n이 포함된 문자열의 컬렉션의 경우에는 writelines를 이용해서 줄 단위 기록 가능
    - ₩n이 포함된 문자열의 컬렉션의 경우에는 join을 이용해서 줄 단위 기록이 가능

파일 처리

- open() 함수
  - ✓ 파일 읽기
    - 파일 경로만 입력하면 읽기 모드로 열림
    - 두번째 매개변수로 'r'을 전달하면 읽기 모드가 되고 생략하고 인코딩 방식을 대입해도 읽기 모드
    - 전체 데이터를 하나의 문자열로 읽고자 하는 경우에는 read()를 이용하면 전체를 읽어옴
    - 줄 단위로 읽을 때는 파일 객체의 반복자 나 readline() 또는 파일 전체를 줄 단위로 끊어서 리스트에 저장하는 readlines 함수를 이용해서 줄 단위로 읽을 수 있음
    - read(정수)를 이용하면 정수만큼의 바이트를 읽어옴

파일 처리

- open() 함수
  - ✓ 파일 읽기

문자	의미
‘r’	읽기용으로 열기 (기본값)
‘w’	쓰기용으로 여는데 같은 경로에 파일이 존재하면 파일 내용을 비움.
‘x’	배타적 생성 모드로 여는데 파일이 존재하면 IOError 예외 발생
‘a’	쓰기용으로 여는데 ‘w’와는 달리 이미 같은 경로에 파일이 존재하는 경우 기존 내용에 덧붙이기를 함
‘b’	바이너리 모드
‘t’	텍스트 모드(기본값)
‘+’	읽기/쓰기용으로 파일 읽기

파일 처리

- 파일의 경로 설정
  - ✓ 절대 경로
    - 파일의 경로를 루트부터 직접 기재하는 방식
    - 리눅스나 매킨토시는 디렉토리 구분 기호로 /를 사용하고 윈도우의 경우에는 ₩을 사용
    - 절대 경로를 사용할 때 윈도우에서는 디렉토리 구분 기호를 ₩₩ 로 설정
    - /로 시작하면 루트 디렉토리
  - ✓ 상대 경로
    - 파일의 경로를 현재 위치로부터의 상대적인 경로로 설정하는 방식
    - /를 이용해서 디렉토리를 구분
    - ./는 현재 디렉토리 – ./ 없이 파일 경로만 입력해도 현재 디렉토리
    - ../는 상위 디렉토리

파일 쓰기

- 작업 디렉토리 변경

```
import os
print (os.getcwd()) #현재 디렉토리 확인

#현재 작업 디렉토리 변경
#os.chdir('/Users/munseokpark/Documents/python/source/basic')
```

파일 쓰기

❖ 파일에 기록하기

```
try:
    file = open('./data/test.txt', 'w')
    #파일 객체 정보 출력
    print(file)
    #데이터를 한번에 기록
    file.write('Hello Python')
    file.write('₩n₩n')
    #데이터를 줄 단위로 기록
    lines = ['안녕하세요', '반갑습니다.','파이썬입니다.']
    file.write('₩n'.join(lines))

    '''
    ₩n이 포함된 경우는
    file.write(''.join(lines))
    file.writelines(lines)
    가능
    '''

    print('기록 성공')
except Exception as e:
    print("파일 처리 중 예외 발생:", e)
finally:
    file.close()
```

기록 성공

파일 읽기

❖ 파일의 내용 읽어오기

```
try:
    file = open('./data/test.txt', 'r')
    #한꺼번에 전부 읽기
    content = file.read()
    print(content)
except Exception as e:
    print("파일 처리 중 예외 발생:", e)
finally:
    file.close()
print("==============")

#줄단위로 읽기
try:
    file = open('./data/test.txt', 'r')
    for line in file:
        print(line)
except Exception as e:
    print("파일 처리 중 예외 발생:", e)
finally:
    file.close()
print("==============")
```

```
Hello Python

안녕하세요
반갑습니다.
파이썬입니다.
Hello Python


안녕하세요

반갑습니다.

파이썬입니다.
==============
['Hello Python₩n', '₩n', '안녕하세요
₩n', '반갑습니다.₩n', '파이썬입니
다.']
```

파일 읽기

- 파일의 내용 읽어오기

```
try:
    file = open('./data/test.txt', 'r')
    lines = file.readlines()
    print(lines)
except Exception as e:
    print("파일 처리 중 예외 발생:", e)
finally:
    file.close()
```

파일 처리

- 파일 관리
  - ✓ 파일이나 소켓과 같은 리소스에 액세스하는 경우에 일반적으로 해당 리소스를 열면서 제어를 위한 핸들을 얻고 리소스의 회수를 위해서 모든 작업이 완료된 경우에는 반드시 해당 핸들을 닫는 식의 흐름이 필요
  - ✓ 리소스의 열기와 닫기 사이에는 예측하지 못한 변수가 발생할 수 있는데 특정한 조건에 의해서나 예외가 발생하는 등의 이유로 리소스의 정리를 못한 채로 루틴이 끝나거나 프로그램이 종료되는 경우가 발생할 수 있음
  - ✓ 이런 경우에는 try/except/finally 구문을 통해서 흐름의 끝에서 리소스를 정리하고 루틴을 떠나는 형태로 코드를 작성하는 방법도 있지만 이러한 방법 역시 예외가 발생하는 케이스와 탈출 조건을 만족하는 케이스에 대해서 각각 리소스를 정리하는 코드가 중복으로 들어가야 하는 경우가 있음
  - ✓ 파이썬의 컨텍스트 매니저는 이러한 리소스의 액세스를 with 라는 구문을 통해서 특정 블럭 내의 동작으로 제한하고 블럭을 나가는 경우에 그 사유가 어떻게 되든 간에 리소스의 해제 처리를 자동으로 하는 것을 보장

```
with open('파일경로', '파일모드', encoding='utf8') as 변수:
```

파일 처리

- 파일 관리
  - ✓ 컨텍스트 매니저 객체들은 with 문과 함께 사용되었을 때 다음과 같은 동작을 처리
    - 객체가 생성된 후 with 블럭에 진입하면서 미리 지정된 특정한 작업을 수행
    - with 블럭을 떠나는 시점에 미리 지정된 특정한 작업을 수행
  - ✓ 시작 지점에는 특정한 동작을 수행하는 일이 별로 없음
  - ✓ 파일 객체의 경우에 with 블록을 떠나는 시점에 자신을 닫는 동작을 수행하도록 미리 정해져 있기 때문에 with 문 내에서 예외가 발생해서 실행이 중단되거나, 함수 내에서 리턴하는 동작을 만나더라도 파일 객체는 닫기를 수행하는 기회를 보장
  - ✓ with 문을 쓸 때의 장점은 위의 예에서 보듯이 with 블럭 자체가 중첩이 가능하다는 점인데 어느 한 지점에서 문제가 발생하여 모든 블럭을 빠져나와야 하는 상황에서도 가장 하위의 블럭에서부터 파일을 순차적으로 닫고 안전하게 리소스를 회수하며 제어 로직에서의 전후 관계를 따로 추적하지 않아도 되어 그만큼 코드가 간결해짐

파일 처리

- CSV
  - ✓ csv란 Comma Separated Values의 약자로서 csv 파일은 각 라인의 컬럼들이 구분자(콤마가 많이 사용되지만 공백 등도 사용됨)로 분리된 텍스트 파일 포맷
  - ✓ 간단한 형태의 csv 파일은 문자열을 콤마로 split 하여 처리하면 되지만 간혹 컬럼 데이터에 콤마가 있는 경우나 이중 인용부호로 감싸서 데이터 내의 콤마를 Escape하기 (예: "Lee, Alex") 때문에 파이썬에 내장된 csv 모듈을 사용하여 csv 파일을 처리하는 것이 좋음
  - ✓ csv 모듈을 이용한 csv 파일 읽기
    - csv 파일을 읽기 위해서는 먼저 파이썬에 기본 내장된 csv 모듈을 import
    - csv 파일을 오픈
    - 파일객체를 csv.reader(파일객체) 에 대입
    - csv.reader() 함수는 Iterator 타입인 reader 객체를 리턴하므로 for 루프를 돌며 한 라인씩 가져올 수 있는데 이 때 리턴되는 각 라인은 컬럼들을 나열한 리스트(list)
  - ✓ csv 파일에 기록하기
    - csv 파일을 쓰기 위해서는 .csv 파일을 쓰기 모드로 오픈
    - 파일객체를 csv.writer(파일객체) 에 대입
    - csvwriter는 writerow() 라는 함수를 통해 list 데이터를 한 라인 추가하게 되는데 윈도우즈의 경우 csv 모듈에서 데이터를 쓸 때 각 라인 뒤에 빈 라인이 추가되는 문제가 있는데 이를 없애기 위해 파일을 open 할 때 newline='' 옵션을 지정
  - ✓ 구분자가 ,가 아닌 경우는 읽거나 기록할 때 delimiter 옵션을 추가

파일 읽기(csv)

- test.csv 읽기

```
with open('./data/test.csv', 'r') as f:
    #한꺼번에 전부 읽기
    content = f.read()
    print("==============")
    ar = content.split(',')
    print(type(ar))
    for imsi in ar:
        print(imsi)
```

```
==============
<class 'list'>
park
kim
choi
```

파일 읽기(csv)

❖ singer.csv 읽기

```
import csv
with open('./data/singer.csv', 'r', encoding='cp949') as f:
    rdr = csv.reader(f)
    for line in rdr:
        print(line)
```

```
['1', '태연', '1989-03-09', '소녀시대']
['2', '재경', '1988-12-24', '레인보우']
['3', '아이린', '1991-03-29', '레드벨벳']
```

파일 읽기(csv)

- singer.csv 쓰기

```
import csv
with open('./data/singer.csv', 'a', encoding='cp949', newline='') as f:
    wr = csv.writer(f)
    wr.writerow([4, "최유정", "1999-11-12", "위키미키"])
    wr.writerow([5, "아이유", "1993-05-16", "솔로"])
```

바이너리 파일

- 바이너리 파일
  - ✓ 바이너리 파일은 바이트 단위로 데이터를 읽고 쓰는 것
  - ✓ 바이너리 파일은 문자열을 기록할 수 없고 byte 단위로만 기록
  - ✓ 문자열의 경우는 encode 함수를 이용해서 byte로 변형 할 수 있고 바이트를 문자열로 변환할 때는 decode 함수를 이용
  - ✓ 파일을 열 때 모드 뒤에 b를 추가

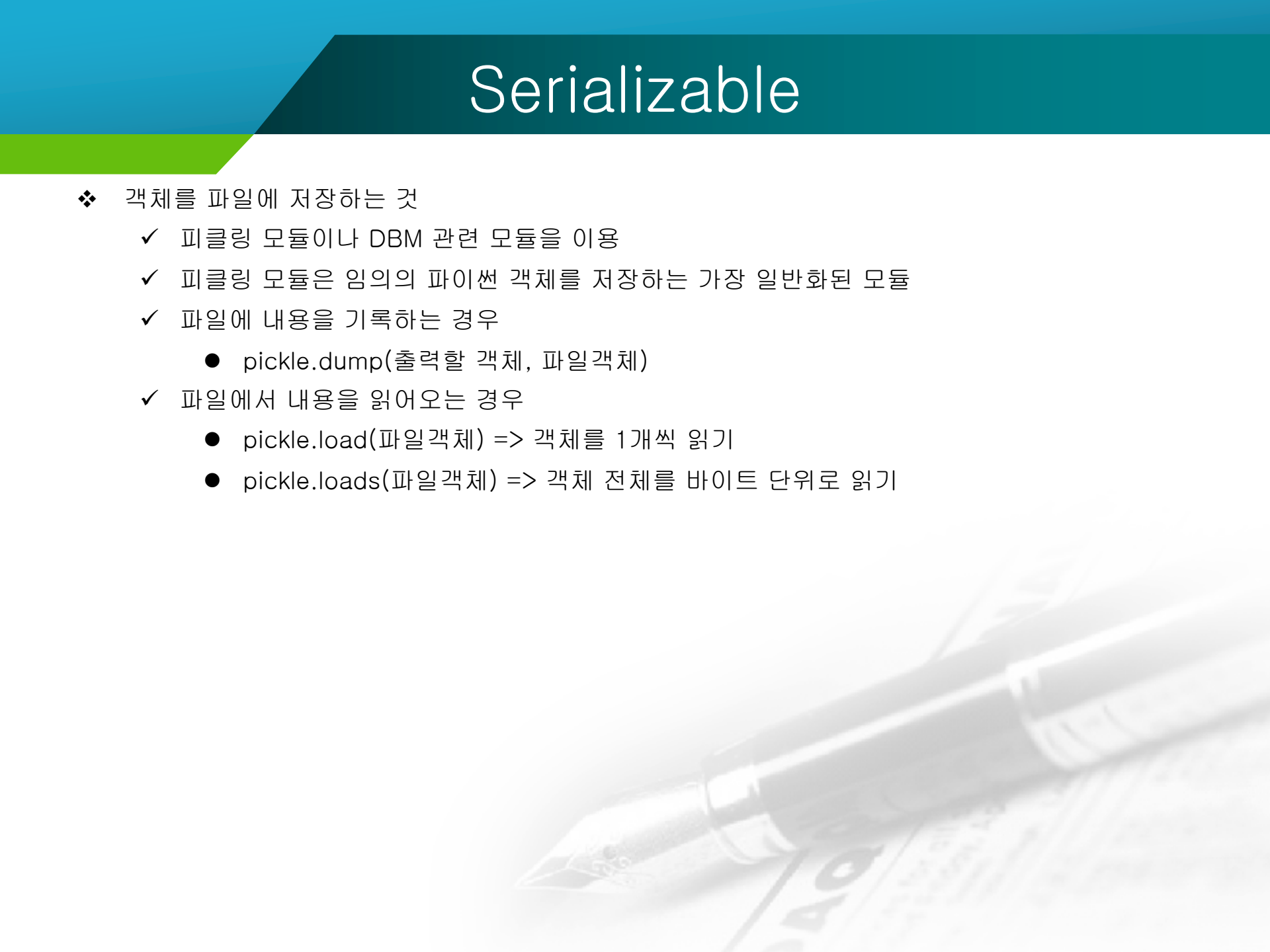
```
with open('data/test.bin', 'wb') as f:
    f.write("안녕하세요".encode())

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

문자열을 바이트 단위로 기록했으므로
읽을 때는 문자열로 변환해서 읽어야 함

Serializable

- 객체를 파일에 저장하는 것
  - ✓ 피클링 모듈이나 DBM 관련 모듈을 이용
  - ✓ 피클링 모듈은 임의의 파이썬 객체를 저장하는 가장 일반화된 모듈
  - ✓ 파일에 내용을 기록하는 경우
    - pickle.dump(출력할 객체, 파일객체)
  - ✓ 파일에서 내용을 읽어오는 경우
    - pickle.load(파일객체) => 객체를 1개씩 읽기
    - pickle.loads(파일객체) => 객체 전체를 바이트 단위로 읽기



Serializable

- 객체 직렬화 – 저장한 인스턴스의 클래스

```
class Dto:
    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 "{번호:" + str(self.num) + ",이름:" + self.name + "}"
```

Serializable

- 객체 직렬화 – 저장한 인스턴스 생성

```
data1 = Dto()
data1.setNum(1)
data1.setName("park")
data2 = Dto()
data2.setNum(2)
data2.setName("kim")
li = [data1, data2]
```

Serializable

❖ 객체 직렬화 – 직렬화

```
import pickle
try:
    with open('./data/test.txt', 'wb') as f:
        pickle.dump(li, f)
        f.close()
    with open('./data/test.txt', 'rb') as f:
        result = pickle.load(f)
        for temp in result:
            print(temp.toString())
        f.close()
except Exception as e:
    print("예외 발생:", e)
finally:
    f.close()
```

{번호:1,이름:park}
{번호:2,이름:kim}

파일 압축

- 파일 압축
  - ✓ zip 파일 압축은 zipfile 모듈 사용
    - ZipFile 이라는 함수로 압축 객체를 생성하고 write 함수를 이용해서 하나씩 압축
    - 압축 해제는 압축 파일을 가지고 ZipFile을 만든 후 extractall()을 호출
  - ✓ tar 파일 압축은 tarfile 모듈 사용
    - open 함수를 이용해서 압축 객체를 만든 후 add 함수를 이용해서 파일을 추가
    - 압축 해제는 압축 파일 이름을 가지고 open 함수로 객체를 생성한 후 extractalll()을 호출

파일 압축

- 파일 압축

```
import zipfile
filelist = ["data/test.txt"]
with zipfile.ZipFile('data/test.zip', 'w', compression=zipfile.ZIP_BZIP2) as myzip:
    for temp in filelist:
        myzip.write(temp)
zipfile.ZipFile('data/test.zip').extractall()
```

파일 압축

- 파일 압축

```
import tarfile
filelist = ["./data/test.txt"]
with tarfile.open('data/test.tar.gz', 'w:gz') as mytar:
    for temp in filelist:
        mytar.add(temp)
tarfile.open('data/test.tar.gz').extractall()
```

로그 파일 읽기

- Apache Web Server 로그 파일

  180.76.15.5 - - [15/Nov/2015:03:45:45 +0000] "GET / HTTP/1.1" 200 6812

  180.76.15.143 - - [15/Nov/2015:03:46:26 +0000] "GET /samsblog/?p=2 HTTP/1.1" 301 -

  - ✓ 공백으로 구분된 10개의 데이터가 한 줄에 존재
  - ✓ 행의 마지막 데이터는 트래픽 인데 없는 경우에는 -

로그 파일 읽기

- Apache Web Server 에서 유효한 페이지 접근 횟수 출력

```
pageviews = 0
f = None
with open('data/log.txt', 'r') as f:
    logs = f.readlines()
    for log in logs:
        log = log.split()
        status = log[8]
        if status == '200':
            pageviews += 1
    print('총 페이지뷰: [%d]' %pageviews)
```

로그 파일 읽기

- Apache Web Server 에서 ip 수 출력

```
visit_ip = []
f = None
with open('data/log.txt', 'r') as f:
    logs = f.readlines()
    for log in logs:
        log = log.split()
        ip = log[0]
        if ip not in visit_ip:
            visit_ip.append(ip)
    print('방문자수: [%d]' %len(visit_ip))
```

로그 파일 읽기

- Apache Web Server 에서 전체 트래픽 출력

```
total_service = 0
with open('data/log.txt', 'r') as f:
    logs = f.readlines()
    for log in logs:
        log = log.split()
        servicebyte = log[9]
        if servicebyte.isdigit():
            total_service += int(servicebyte)
    total_service /= 1024
    print('총 서비스 용량: %dKB' %total_service)
```

로그 파일 읽기

- Apache Web Server 에서 ip 별 트래픽 출력

```
services = {}
f = None
with open('data/log.txt', 'r') as f:
    logs = f.readlines()
    for log in logs:
        log = log.split()
        ip = log[0]
        servicebyte = log[9]
        if servicebyte.isdigit():
            servicebyte = int(servicebyte)
        else:
            servicebyte = 0
        if ip not in services:
            services[ip] = servicebyte
        else:
            services[ip] += servicebyte
    ret = sorted(services.items(), key=lambda x: x[1], reverse=True)
    print('사용자IP - 서비스용량')
    for ip, b in ret:
        print('[%s] - [%d]' %(ip, b))
```

엑셀 API

- 엑셀 API
  - ✓ 파이썬에서 엑셀 사용하기 – 파이썬에서 엑셀 데이터를 핸들링하기 위해서는 openpyxl, xlrd, xlrw 등의 외부 패키지를 설치해서 사용
  - ✓ openpyxl 패키지

    pip install openpyxl
  - ✓ 엑셀 데이터 읽고 쓰기
    - 엑셀 파일을 오픈하기 위해 openpyxl.load_workbook(엑셀파일명) 함수를 호출하여 Workbook 객체를 생성
    - 하나의 Workbook에는 여러 개의 Worksheet 들이 있음
    - Worksheet 객체의 active 를 통해 현재 활성화 된 워크시트를 가져올 수 있고 wb.get_sheet_by_name(시트명)을 이용해서 시트명에 해당하는 시트의 데이터를 가져올 수 있음
    - 워크시트는 행(Row)과 열(Column)로 구성되어 있는데 시트 내에 데이터가 있는 부분의 행들은 시트객체.rows 를 통해 엑세스 할 수 있고 마찬가지로 시트객체.columns는 유효 컬럼들을 엑세스하는데 사용
    - 특정 cell 에 값을 저장하기 위해서는 cell.value 에 값을 넣으면 되는데, 시트에서 cell 을 지정하기 위해 ws["A1"]과 같이 엑셀식 cell 지정법을 사용할 수 있고, 또한 행과 열 인덱스를 사용하여 ws.cell(row=행인덱스, column=열인덱스) 표현을 사용 가능
    - 엑셀의 변경 내용을 저장하기 위해서는 Workbook 객체에서 save() 함수를 사용하며 엑셀 사용이 모두 끝난 경우 close() 함수를 호출

엑셀 API

name	국어	영어	수학
주시현	89	93	98
최경우	95	87	73
정정원	85	89	79

```
========== RESTART: /
주시현 : 89 93 98 280
최경우 : 95 87 73 255
정정원 : 85 89 79 253
```

엑셀 API

- 엑셀 파일 읽기 와 쓰기

```
import openpyxl
wb = None
try:
    # 엑셀파일 열기
    wb = openpyxl.load_workbook('data/score.xlsx')

    # 현재 Active Sheet 얻기
    ws = wb.active
    # ws = wb.get_sheet_by_name("Sheet1“)

    for r in ws.rows:
        row_index = r[0].row   # 행 인덱스
        if row_index == 1:
            ws.cell(row=row_index, column=5).value = '합계'
            continue
        name = r[0].value
        kor = r[1].value
        eng = r[2].value
        math = r[3].value
        sum = kor + eng + math
```

엑셀 API

❖ 엑셀 파일 읽기 와 쓰기

```
            # 합계 쓰기
            ws.cell(row=row_index, column=5).value = sum
            print(name, ':' , kor, eng, math, sum)
        # 엑셀 파일 저장
        wb.save("data/score2.xlsx")
    except Exception as e:
        print("예외 발생:", e)
    finally:
        wb.close()
```

주시현 : 89 93 98 280
최경우 : 95 87 73 255
정정원 : 85 89 79 253