

데이터 과학을 위한 파이썬 프로그래밍



13. CSV와 로그 관리

목차

1. CSV
2. 로그 관리
3. 설정 저장
4. Lab: 로깅 프로그램

01

CSV

01. CSV

■ CSV의 개념

- CSV(Comma Separate Values)는 콤마(,)를 기준으로 나누어진 값이라고 이해하면 된다.
- csv 파일은 어디서나 사용할 수 있도록 텍스트 데이터를 사용한다.

연도, 제조사, 모델, 설명, 가격

1997, Ford, E350, "ac, abs, moon", 3000.00

1999, Chevy, "Venture ""Extended Edition""", "", 4900.00

1999, Chevy, "Venture ""Extended Edition, Very Large""", "", 5000.00

1996, Jeep, Grand Cherokee, "MUST SELL! air, moon roof, loaded", 4799.00

- ➡ 위 데이터를 보면, 제일 상단에 필드(field), 헤더(header) 또는 열 이름(column name)이라고 부르는 텍스트 데이터가 입력되었다. 각 데이터는 콤마로 나뉘어진다. 두 번째 줄부터는 각 필드의 실제 데이터가 있는데, 각 행의 데이터가 인스턴스라고 이해하면 된다. 이러한 데이터를 행(row), 튜플(tuple), 인스턴스(instance) 등으로 부른다. 데이터의 분류는 분류 기준이 되는 문자에 따라 TSV(Tab Separate Values), SSV(Single-blank Separate Values) 등으로 구분한다.

01. CSV

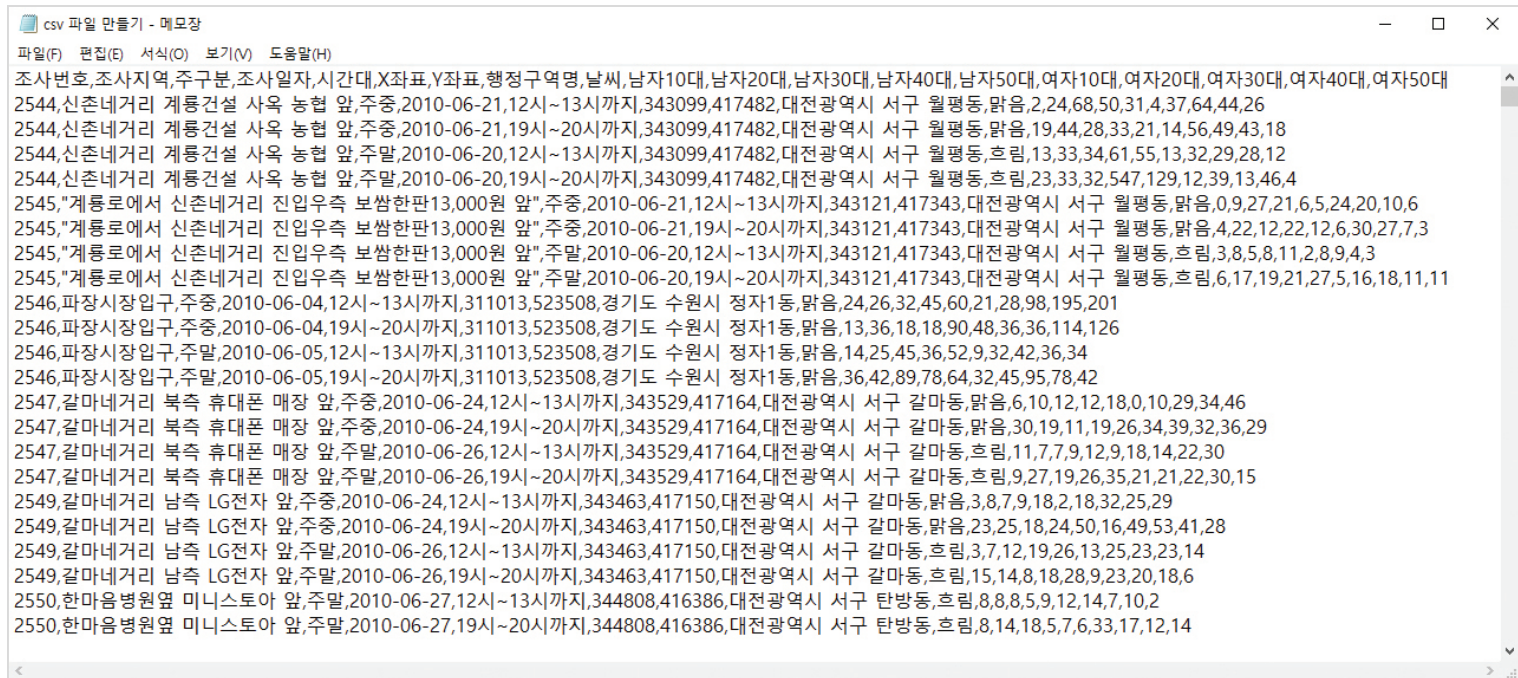
■ csv 파일 만들기

- csv 파일은 엑셀을 사용하여 간단히 만들 수 있다. 다음 순서대로 csv 파일을 만들어 보자.
 - ① 소스 파일에서 'csv 파일 만들기.xls' 파일을 다운로드한다.
 - ② 엑셀에서 다운로드한 파일을 연다.
 - ③ 메뉴 바에서 [파일]-[다른 이름으로 저장]을 선택한다.
 - ④ [다른 이름으로 저장] 대화상자에서 '파일 형식'을 'CSV(쉼표로 분리)'로 선택한 후, [저장]을 클릭한다.
 - ⑤ 엑셀을 종료한 후, 메모장에서 파일을 연다.

01. CSV

■ csv 파일 만들기

- 새롭게 생성된 'csv 파일 만들기.csv' 파일은 메모장 같은 텍스트 에디터로 내용을 확인할 수 있다.



[새로 만든 csv 파일을 메모장에서 확인하기]

01. CSV

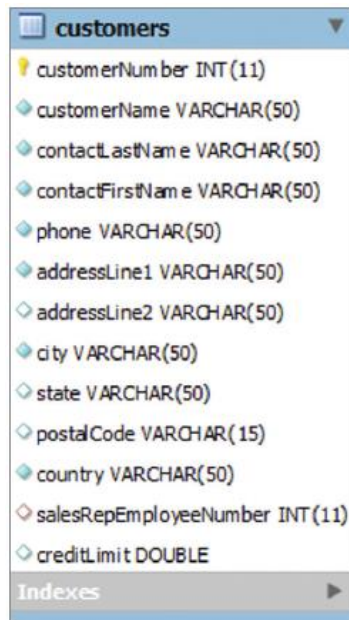
■ csv 파일 다루기

- csv 파일을 다루기 위해 먼저 파이썬으로 csv 파일을 열고, 내용을 사용하는 작업을 한다. 파이썬으로 csv 파일을 다루는 방법에는 크게 세 가지가 있다.
 - ① 파일 객체 사용하기
 - ② csv 객체 사용하기
 - ③ PANDAS 객체 사용하기

01. CSV

■ csv 파일 다루기 : 파일 객체를 사용하여 데이터 다루기

- 파일 객체를 사용하여 데이터를 다루는 방법은 일반적인 텍스트 파일을 처리하듯 파일을 읽어 온 후, 한 줄씩 데이터를 처리하는 것이다.
- 소스 파일로 제공하는 'customers.csv' 파일을 사용하여 실습한다.



(a) 데이터 구조



(b) 데이터 형태

[customers.csv의 데이터 구조와 데이터 형태]

01. CSV

■ csv 파일 다루기 : 파일 객체를 사용하여 데이터 다루기

코드 13-1 csv1.py

```
1 line_counter = 0           # 파일의 총 줄 수를 세는 변수
2 data_header = []          # 데이터의 필드값을 저장하는 리스트
3 customer_list = []         # customer의 개별 리스트를 저장하는 리스트
4
5 with open("customers.csv") as customer_data: # customers.csv 파일을 customer_
                                                data 객체에 저장
6     while 1:
7         data = customer_data.readline() #customers.csv의 데이터 변수에 한 줄씩 저장
8         if not data:break               #데이터가 없을 때, 반복문 종료
9         if line_counter == 0:           #첫 번째 데이터는 데이터의 필드
10            data_header = data.split(",") # 데이터의 필드는 data_header 리스트
                                                에 저장, 데이터 저장 시 ","로 분리
11        else:
12            customer_list.append(data.split(",")) # 일반 데이터는 customer_list
                                                객체에 저장, 데이터 저장 시 ","
                                                로 분리
```

01. CSV

■ csv 파일 다루기 : 파일 객체를 사용하여 데이터 다루기

```
13         line_counter +=1
14
15     print("Header:", data_header)           # 데이터 필드값 출력
16     for i in range(0, 10):                  # 데이터 출력(샘플 10개만)
17         print("Data",i,":",customer_list[i])
18     print(len(customer_list))               # 전체 데이터 크기 출력
```

```
Header: ['customerNumber', 'customerName', 'contactLastName', 'contactFirstName',
'phone', 'addressLine1', 'addressLine2', 'city', 'state', 'postalCode', 'country',
'salesRepEmployeeNumber', 'creditLimit\n']
Data 0 : ['103', 'Atelier graphique', 'Schmitt', 'Carine ', '40.32.2555', '"54', '
rue Royale"', 'NULL', 'Nantes', 'NULL', '44000', 'France', '1370', '21000\n']
Data 1 : ['112', 'Signal Gift Stores', 'King', 'Jean', '7025551838', '8489 Strong
St.', 'NULL', 'Las Vegas', 'NV', '83030', 'USA', '1166', '71800\n']
:      ← Data 2부터 Data 9까지 생략
122
```

01. CSV

■ csv 파일 다루기 : 파일 객체를 사용하여 데이터 다루기

- ➡ 이 코드에서는 `readline()` 함수와 `split()` 함수가 핵심적으로 사용된다. 1~3행에서는 사용할 변수와 리스트를 지정하고, 5행에서는 `open()` 함수를 사용하여 해당 파일을 연다. 7행에서 `readline()` 함수를 사용하여 파일의 내용을 한 줄씩 읽어 온다. `line_counter` 변수를 사용하여 줄 수를 세면서 결과값 첫 번째 줄에는 데이터의 헤더가 들어가고, 다음 줄부터 데이터를 가져와 `split()` 함수로 잘라 주며 `customer_list` 변수를 `append()` 함수로 추가한다.
- ➡ 직관적으로 이해할 수 있는 매우 쉬운 구조이다. 일반적으로 csv 파일의 첫 줄에 다른 엑셀 파일처럼 각 데이터의 제목인 헤더나 열 이름이 있으므로 따로 처리한다. 데이터는 [코드 13-1]과 같이 리스트 형태로 받을 수 있지만, 열 이름을 키의 값으로 하여, 딕셔너리형으로 처리할 수도 있다.

01. CSV

■ csv 파일 다루기 : 파일 객체를 사용하여 데이터 다루기

- 이번에는 읽은 정보를 csv 파일에 쓰는 실습이다. 비슷하지만, 필요한 정보만 따로 리스트 객체에 저장하여 csv 파일에 쓰는 방식으로 작성할 수 있다.

코드 13-2 csv2.py

```
1 line_counter = 0
2 data_header = []
3 employee = []
4 customer_USA_only_list = []
5 customer = None
6
7 with open("customers.csv", "r") as customer_data:
8     while 1:
9         data = customer_data.readline()
10        if not data:
11            break
12        if line_counter == 0:
13            data_header = data.split(",")
14        else:
```

01. CSV

■ csv 파일 다루기 : 파일 객체를 사용하여 데이터 다루기

```
15         customer = data.split(",")
16
17         if customer[10].upper()=="USA": # customer 데이터의 offset 10번째 값
18             customer_USA_only_list.append(customer)
                                     # 즉, country 필드가 "USA" 것만
                                     # customer_USA_only_list에 저장
19
19         line_counter+=1
20
21     print("Header:", data_header)
22     for i in range(0, 10):
23         print("Data:",customer_USA_only_list[i])
24     print(len(customer_USA_only_list))
25
26     with open("customers_USA_only.csv","w") as customer_USA_only_csv:
27         for customer in customer_USA_only_list:
28             customer_USA_only_csv.write(",".join(customer).strip('\n')+"\n")
            # customer_USA_only_list 객체에 있는 데이터를 customers_USA_only.csv 파일에 쓰기
```

01. CSV

■ csv 파일 다루기 : 파일 객체를 사용하여 데이터 다루기

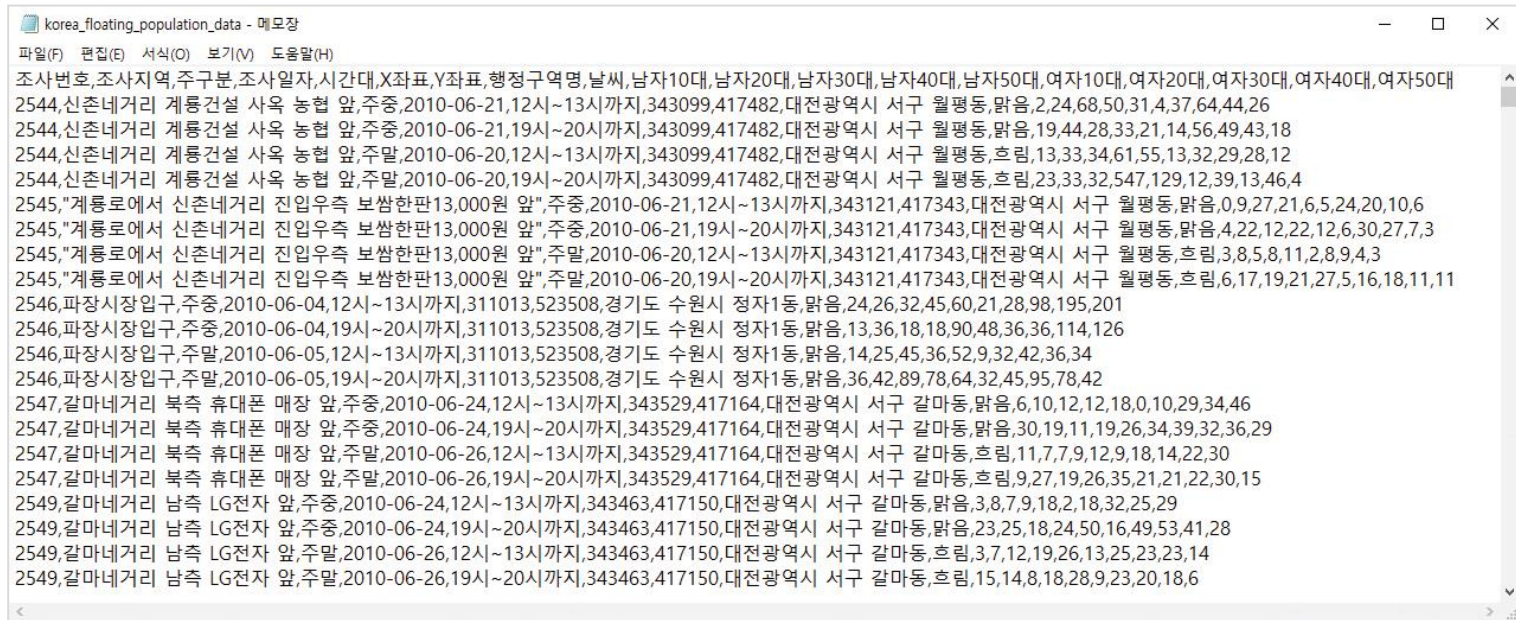
```
Header: ['customerNumber', 'customerName', 'contactLastName', 'contactFirstName',  
'phone', 'addressLine1', 'addressLine2', 'city', 'state', 'postalCode', 'country',  
'salesRepEmployeeNumber', 'creditLimit\n']  
Data: ['112', 'Signal Gift Stores', 'King', 'Jean', '7025551838', '8489 Strong  
St.', 'NULL', 'Las Vegas', 'NV', '83030', 'USA', '1166', '71800\n']  
Data: ['124', 'Mini Gifts Distributors Ltd.', 'Nelson', 'Susan', '4155551450',  
'5677 Strong St.', 'NULL', 'San Rafael', 'CA', '97562', 'USA', '1165', '210500\n']  
:  
← 생략  
34
```

- ➔ [코드 13-2]에서는 각 데이터의 10번째 열, 즉 customer[10]이 USA인지 확인한다. 10번째 열은 각 customers의 국가 정보를 입력하는 곳으로, 미국 고객의 정보만 따로 추출하기 위해 사용할 수 있다. 추출된 정보는 customer_USA_only_list라는 리스트 객체에 저장하여 사용된다. 그리고 모든 파일 읽기 코드가 실행된 후 파일 쓰기 코드를 실행하여 해당 내용을 'customers_USA_only.csv'에 작성한다. 파일을 작성할 때는 콤마를 사용하여 데이터를 잘라주는 ", ".join(customer)를 넣어야 하며, 각 데이터의 끝에 줄 바꿈 정보도 넣어야 한다.

01. CSV

■ csv 파일 다루기 : csv 객체를 사용하여 데이터 다루기

- 파이썬에서는 기본적으로 파일 객체가 아닌 csv 객체를 이용하여 csv 파일 형태의 데이터를 다루기 쉽다.
- 이번 실습에서는 소스 파일로 제공하는 'korea_floating_population_data.csv' 파일을 사용한다.



```
korea_floating_population_data - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
조사번호,조사지역,주구분,조사일자,시간대,X좌표,Y좌표,행정구역명,날씨,남자10대,남자20대,남자30대,남자40대,남자50대,여자10대,여자20대,여자30대,여자40대,여자50대
2544,신촌네거리 계통건설 사옥 농협 앞,주중,2010-06-21,12시~13시까지,343099,417482,대전광역시 서구 월평동,맑음,2,24,68,50,31,4,37,64,44,26
2544,신촌네거리 계통건설 사옥 농협 앞,주중,2010-06-21,19시~20시까지,343099,417482,대전광역시 서구 월평동,맑음,19,44,28,33,21,14,56,49,43,18
2544,신촌네거리 계통건설 사옥 농협 앞,주말,2010-06-20,12시~13시까지,343099,417482,대전광역시 서구 월평동,흐림,13,33,34,61,55,13,32,29,28,12
2544,신촌네거리 계통건설 사옥 농협 앞,주말,2010-06-20,19시~20시까지,343099,417482,대전광역시 서구 월평동,흐림,23,33,32,547,129,12,39,13,46,4
2545,"계룡로에서 신촌네거리 진입우측 보쌈한판13,000원 앞",주중,2010-06-21,12시~13시까지,343121,417343,대전광역시 서구 월평동,맑음,0,9,27,21,6,5,24,20,10,6
2545,"계룡로에서 신촌네거리 진입우측 보쌈한판13,000원 앞",주중,2010-06-21,19시~20시까지,343121,417343,대전광역시 서구 월평동,맑음,4,22,12,22,12,6,30,27,7,3
2545,"계룡로에서 신촌네거리 진입우측 보쌈한판13,000원 앞",주말,2010-06-20,12시~13시까지,343121,417343,대전광역시 서구 월평동,흐림,3,8,5,8,11,2,8,9,4,3
2545,"계룡로에서 신촌네거리 진입우측 보쌈한판13,000원 앞",주말,2010-06-20,19시~20시까지,343121,417343,대전광역시 서구 월평동,흐림,6,17,19,21,27,5,16,18,11,11
2546,파장시장입구,주중,2010-06-04,12시~13시까지,311013,523508,경기도 수원시 정자1동,맑음,24,26,32,45,60,21,28,98,195,201
2546,파장시장입구,주중,2010-06-04,19시~20시까지,311013,523508,경기도 수원시 정자1동,맑음,13,36,18,18,90,48,36,36,114,126
2546,파장시장입구,주말,2010-06-05,12시~13시까지,311013,523508,경기도 수원시 정자1동,맑음,14,25,45,36,52,9,32,42,36,34
2546,파장시장입구,주말,2010-06-05,19시~20시까지,311013,523508,경기도 수원시 정자1동,맑음,36,42,89,78,64,32,45,95,78,42
2547,갈마네거리 북측 휴대폰 매장 앞,주중,2010-06-24,12시~13시까지,343529,417164,대전광역시 서구 갈마동,맑음,6,10,12,12,18,0,10,29,34,46
2547,갈마네거리 북측 휴대폰 매장 앞,주중,2010-06-24,19시~20시까지,343529,417164,대전광역시 서구 갈마동,맑음,30,19,11,19,26,34,39,32,36,29
2547,갈마네거리 북측 휴대폰 매장 앞,주말,2010-06-26,12시~13시까지,343529,417164,대전광역시 서구 갈마동,흐림,11,7,7,9,12,9,18,14,22,30
2547,갈마네거리 북측 휴대폰 매장 앞,주말,2010-06-26,19시~20시까지,343529,417164,대전광역시 서구 갈마동,흐림,9,27,19,26,35,21,21,22,30,15
2549,갈마네거리 남측 LG전자 앞,주중,2010-06-24,12시~13시까지,343463,417150,대전광역시 서구 갈마동,맑음,3,8,7,9,18,2,18,32,25,29
2549,갈마네거리 남측 LG전자 앞,주중,2010-06-24,19시~20시까지,343463,417150,대전광역시 서구 갈마동,맑음,23,25,18,24,50,16,49,53,41,28
2549,갈마네거리 남측 LG전자 앞,주말,2010-06-26,12시~13시까지,343463,417150,대전광역시 서구 갈마동,흐림,3,7,12,19,26,13,25,23,23,14
2549,갈마네거리 남측 LG전자 앞,주말,2010-06-26,19시~20시까지,343463,417150,대전광역시 서구 갈마동,흐림,15,14,8,18,28,9,23,20,18,6
```

[korea_floating_population_data.csv의 데이터 형태]

01. CSV

■ csv 파일 다루기 : csv 객체를 사용하여 데이터 다루기

- csv 객체를 열기 위해서는 먼저 import문을 사용하여 csv 객체를 부르고, reader() 함수를 열어 각 속성(attribute)에 값을 넣으면 된다.

코드 13-3 csv3.py

```
1 import csv
2 f = open("./korea_floating_population_data.csv", "r")
3 reader = csv.reader(
4     f,                                # 연결할 대상 파일 객체
5     delimiter=',',                  # 데이터를 분리하는 기준
6     quotechar='"',                  # 데이터를 묶을 때 사용하는 문자
7     quoting=csv.QUOTE_ALL)          # 데이터를 묶는 기준
```

- ➡ [코드 13-3]의 reader() 함수는 비교적 이해하기 쉽다. 4행에서 오픈할 파일 객체를 인수로 넘기는데, 여기서는 f 변수를 넣는다.

01. CSV

■ csv 파일 다루기 : csv 객체를 사용하여 데이터 다루기

➔ 5행에서 delimiter라는 각 데이터를 구분하는 기준을 입력하는데, 콤마(,), \t, -, /, 세미콜론 (;) 등을 구분 기호로 넣을 수 있으며, 데이터 종류에 따라 사용한다.

6행에는 데이터를 묶는 기준인 quotechar를 작성하는데, 때로는 데이터 사이에 콤마(,)처럼 데이터를 나누는 기준 문자열이 들어갈 수도 있다. 예를 들어, 트위터 데이터라고 생각하면 텍스트 데이터에 '+ 텍스트'가 들어 있는 것은 매우 흔하다. 그러므로 이러한 데이터들은 묶어야 하는데, "나 " 또는 | 등으로 묶어 데이터를 처리한다. 데이터를 쉽게 묶기 위해서는 잘 쓰지 않는 텍스트들이 csv에 사용될 수도 있다.

7행의 quoting은 데이터를 묶는 기준을 어떻게 정할 것인지 결정한다. quoting의 기본 속성은 csv에 다음과 같이 정의되어 있다. 기본 설정은 csv.QUOTE_MINIMAL로 설정된다.

- **QUOTE_ALL:** 모든 데이터를 자료형에 상관없이 묶는다. 모든 데이터를 문자열형으로 처리한다.
- **QUOTE_MINIMAL:** 최소한의 데이터만 묶는다. 예를 들어, ',' 같은 데이터가 포함된 데이터만 묶는다.
- **QUOTE_NONNUMERIC:** 숫자 데이터가 아닌 경우에만 묶는다. 이 경우, 데이터를 읽어 올때 묶이지 않은 데이터는 csv 객체에 의해 실수형으로 읽어 오게 된다.
- **QUOTE_NONE:** 데이터를 묶는 작업을 하지 않는다.

01. CSV

■ csv 파일 다루기 : csv 객체를 사용하여 데이터 다루기

- 데이터를 csv로 직접 분석한 코드이다.

코드 13-4 csv4.py

```
1 import csv                # csv 객체 호출
2
3 seoung_nam_data = []      # 기본 변수명 선언
4 header = []
5 rownum = 0
6
7 with open("korea_floating_population_data.csv", "r", encoding = "cp949") as p_
   file:                    # 불러들일 데이터를 선언함, 한글 처리를 위한 'cp949'
8     csv_data = csv.reader(p_file)    # csv 객체를 이용해 csv_data 읽기, 특별히
                                       데이터를 나누는 기준을 정하지 않음
9     for row in csv_data:            # 읽어 온 데이터를 한 줄씩 처리
10         if rownum == 0:
11             header = row            # 첫 번째 줄은 데이터 필드로 따로 저장
12             location = row[7]       # '행정구역' 필드 데이터 추출
```

01. CSV

■ csv 파일 다루기 : csv 객체를 사용하여 데이터 다루기

```
13         if location.find(u"성남시")!=-1:
14             seoung_nam_data.append(row)    # '행정구역' 데이터에 성남시가 있으면
                                             seoung_nam_data List에 추가
15         rownum +=1
16
17     with open("seoung_nam_floating_population_data.csv", "w", encoding="utf8") as
        s_p_file:
18         writer = csv.writer(s_p_file, delimiter='\t', quotechar="\"", quoting=csv.
        QUOTE_ALL)    # csv.writer를 사용해 csv 파일 만들기, delimiter는 필드 구분자,
                     # quotechar는 필드 각 데이터를 묶는 문자, quoting은 묶는 범위
19         writer.writerow(header)            # 제목 필드 파일에 쓰기
20         for row in seoung_nam_data:
21             writer.writerow(row)           # seoung_nam_data의 정보를 리스트에 쓰기
```

01. CSV

■ csv 파일 다루기 : csv 객체를 사용하여 데이터 다루기

➔ [코드 13-4]는 유동인구 데이터에서 성남시의 데이터만 따로 뽑아 csv 파일을 만드는 코드이다.

7행을 보면 파일을 열 때, encoding을 cp949로 지정한다.

8행에서 csv.reader()를 통해 파일 객체인 p_file을 사용하여 데이터를 읽어 주는 것을 확인할 수 있다. 특별히 속성을 지정하지 않은 것은 해당 데이터가 따로 콤마(,)를 기준으로 나누어져 있고, 다른 속성이 지정되어 있기 때문이다. csv 객체를 읽어 오면 데이터가 기본적으로 시퀀스 자료형으로 만들어져 있어 리스트처럼 다룰 수 있다.

9행의 for row in csv_data:는 각 줄의 데이터를 리스트 형태로 가져온다. 목적에 따라 12행과 14행처럼 특정 위치의 데이터에 성남시가 있는지 확인하고, 확인된 데이터를 따로 뽑아 seoung_nam_data 변수에 넣는다.

데이터를 모두 뽑은 후, 새로운 파일을 만드는 코드를 작성하고 18행과 같이 csv.writer() 함수를 사용하여 새로운 writer 객체를 만든다. reader 객체와 마찬가지로 19행과 같이 writer.writerow(header) 코드를 사용해 한 줄의 데이터를 쓸 수 있다. 이 경우 19행과 21행에서는 header와 row와 같은 리스트 형태의 데이터를 가지고 csv 파일을 만들 수 있다.

02

로그 관리

02. 로그 관리

■ 로깅의 개념

- 이동이나 클릭 등 프로그램을 사용할 때 하는 모든 기본적인 이벤트(event)를 저장하는 것을 로그(log)정보를 저장한다고 한다.
- 프로그램이 실행되는 동안 일어나는 정보를 파일이나 기록으로 남기는 일을 로깅(logging)이라고 한다.
- 로그를 기록하는 가장 일반적인 방법은 파일을 생성하여 로그 정보를 남기는 것이다. 처음 프로그램을 실행할 때 로그 파일 하나를 생성하고, 그 후에 발생하는 이벤트를 로그 파일에 저장하는 방식이다.

02. 로그 관리

■ 기본 로그 관리 모듈: logging

- 파이썬의 기본 로그 관리 모듈은 logging 모듈이다.

코드 13-5 logging1.py

```
1 import logging
2
3 logging.debug("틀렸잖아!")
4 logging.info("확인해!")
5 logging.warning("조심해!")
6 logging.error("에러 났어!!!")
7 logging.critical("망했다...")
```

```
WARNING:root:조심해!
ERROR:root:에러 났어!!!
CRITICAL:root:망했다...
```

02. 로그 관리

■ 기본 로그 관리 모듈: logging

- 파이썬 로깅을 실행할 때 로깅 레벨(logging level)을 지정할 수 있다. 이 로깅 레벨은 DEBUG, INFO, WARNING, ERROR, CRITICAL 등 총 5단계로 나눈다.

| 단계 | 개요 | 예시 |
|----------|---|--|
| DEBUG | 개발 시 처리를 기록하는 로그 정보를 남김 | <ul style="list-style-type: none">• 다음 함수로 A를 호출함• 변수 A를 ○○으로 변경함 |
| INFO | 처리가 진행되는 동안의 정보를 알림 | <ul style="list-style-type: none">• 서버가 시작되었음• 서버가 종료됨• 사용자 A가 프로그램에 접속함 |
| WARNING | 사용자가 잘못 입력한 정보나 처리는 가능하지만 의도치 않은 정보가 들어 왔을 때 알림 | <ul style="list-style-type: none">• 문자열 입력을 기대했으나, 정수형이 입력됨, 문자열 casting으로 처리함• 함수에 인수로 이차원 리스트를 기대했으나, 일차원 리스트가 들어옴, 이차원으로 변환 후 처리 |
| ERROR | 잘못된 처리로 에러가 발생하였지만, 프로그램은 동작할 수 있음을 알림 | <ul style="list-style-type: none">• 파일에 기록해야 하는데 파일이 없음, 예외 처리 후 사용자에게 알림• 외부 서비스와 연결 불가 |
| CRITICAL | 잘못된 처리로 데이터가 손실되었거나 프로그램이 더는 동작할 수 없음을 알림 | <ul style="list-style-type: none">• 잘못된 접근으로 해당 파일이 삭제됨• 사용자에 의한 강제 종료 |

02. 로그 관리

■ 기본 로그 관리 모듈: logging

- 파이썬에서 로깅을 사용하기 위해서는 Logger 객체를 활용해야 한다.

코드 13-6 logging2.py

```
1 import logging
2
3 logger = logging.getLogger("main")           # Logger 선언
4 stream_handler = logging.StreamHandler()      # Logger의 출력 방법 선언
5 logger.addHandler(stream_handler)            # Logger의 출력 등록
```

- ➡ 먼저 3행에서 Logger 객체를 획득하고, 4행에서 출력 방법을 선택한다. 출력하는 공간은 파일이나 화면, 다른 네트워크 등을 선택할 수 있는데, 이를 StreamHandler라고 한다. StreamHandler의 역할은 출력 공간 지정이다. 마지막으로 5행에서 해당 핸들러(handler)를 이미 만든 Logger 객체에 추가하는 작업으로 로깅 준비는 끝난다.

02. 로그 관리

■ 기본 로그 관리 모듈: logging

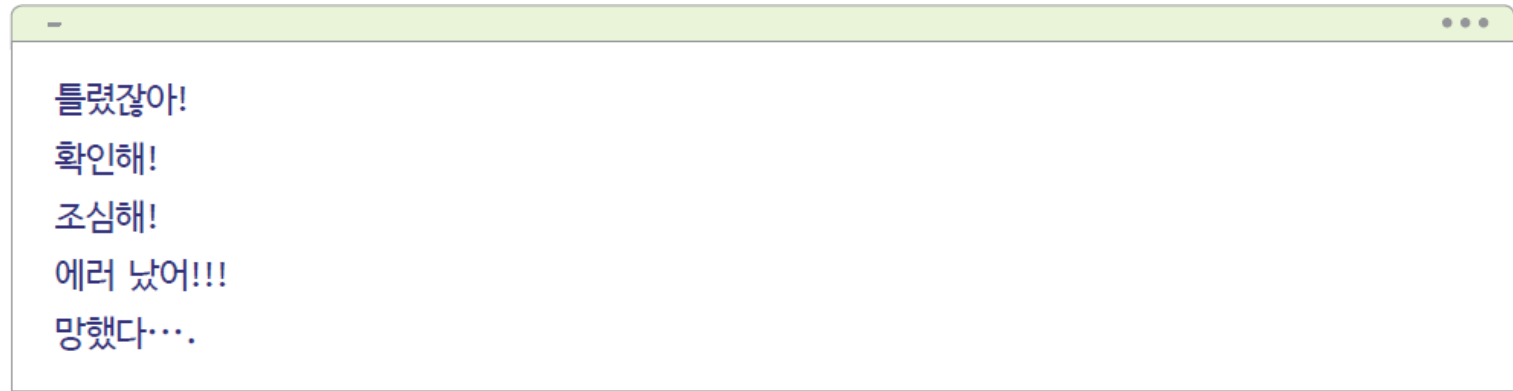
- [코드 13-7]은 해당 Logger가 어느 레벨을 사용해 출력할 것인가를 지정하는 코드이다. 1행에서 DEBUG 레벨을 지정하였으므로 모든 레벨의 로깅 정보를 출력한다.

코드 13-7 logging3.py

```
1 import logging
2
3 logger = logging.getLogger("main")
4 stream_handler = logging.StreamHandler()
5 logger.addHandler(stream_handler)
6
7 logger.setLevel(logging.DEBUG)
8 logger.debug("틀렸잖아!")
9 logger.info("확인해!")
10 logger.warning("조심해!")
11 logger.error("에러 났어!!!")
12 logger.critical("망했다...")
```

02. 로그 관리

■ 기본 로그 관리 모듈: logging



02. 로그 관리

■ 기본 로그 관리 모듈: logging

- [코드 13-8]과 같이 CRITICAL만 출력으로 지정하면 CRITICAL 정보인 '망했다....'만 화면에 출력된다.

코드 13-8 logging4.py

```
1 import logging
2
3 logger = logging.getLogger("main")
4 stream_handler = logging.StreamHandler()
5 logger.addHandler(stream_handler)
6
7 logger.setLevel(logging.CRITICAL)
8 logger.debug("틀렸잖아!")
9 logger.info("확인해")
10 logger.warning("조심해!")
11 logger.error("에러 났어!!!")
12 logger.critical("망했다....")
```



망했다....

03

설정 저장

03. 설정 저장

■ 설정 저장이 필요한 이유

- 어떤 프로그램을 사용할 때 기본 설정을 저장하고, 프로그램을 사용할 때마다 설정된 형태로 프로그램이 실행된다. 따라서 파이썬에서도 설정을 미리 저장해야 한다.

03. 설정 저장

■ 파이썬에서의 설정 저장

- configparser와 argparse는 설정 저장을 파이썬에서 수행할 수 있도록 지원하는 모듈이다. configparser는 설정 자체를 저장하는 것으로, 실행 시점에 설정이 저장된 파일을 읽어 설정을 적용하는 기능을 제공한다. argparse는 configparser와 달리, 실행 시점의 설정 변수들을 직접 지정한다.

03. 설정 저장

■ configparser 모듈

- configparser 모듈은 프로그램의 실행 설정값을 어떤 특정 파일에 저장하여 사용하는 방식이다. 딕셔너리와 비슷하게 설정 파일 안에 키와 값을 넣고, 이를 호출하여 사용한다. 먼저 설정 파일 'example.cfg'를 확인한다.

```
[SectionOne]
Status: Single
Name: Derek
Value: Yes
Age: 30
Single: True

[SectionTwo]
FavoriteColor = Green

[SectionThree]
FamilyName: Johnson
```


03. 설정 저장

■ configparser 모듈

- 위 파일에서 다양한 설정값이 콜론(:)을 통해 키와 값으로 나누어졌다. 그리고 각 값의 상단에는 값의 그룹을 나타내는 섹션이 있다. 섹션은 대괄호로 표현하며, 하위 변수의 묶음을 표현할 때 사용한다. 위에서는 SectionOne, SectionTwo, SectionThree 등 총 3개의 섹션이 있고, 각 섹션 하단에는 1개 이상의 변수가 있다. 만약 'example.cfg'라는 설정 파일이 있다고 하면 이를 사용하는 코드는 다음과 같다.

```
>>> import configparser          #1 configparser 모듈 호출
>>> config = configparser.ConfigParser()  #2 configparser에서 ConfigParser 객체 생성
>>> config.sections()             #3 section 정보 읽어오기
[]
>>> config.read('example.cfg')      #4 특정 파일 안에 있는 설정 정보 읽어오기
['example.cfg']
>>> config.sections()             #5 해당 파일의 section 정보 읽어오기
['SectionOne', 'SectionTwo', 'SectionThree']
>>>
>>> for key in config['SectionOne']:  #6 'SectionOne'에 있는 키 출력
```

03. 설정 저장

■ configparser 모듈

```
...     print(key)
...
status
name
value
age
single
>>> config['SectionOne']['status']      #7 'SectionOne'의 'status' 키의 값 출력
'Single'
```

- ➡ 먼저 #1에서 configparser 모듈을 호출하고, #2에서 ConfigParser 객체를 생성하여 사용한다. #3에서는 해당 객체에서 section 정보를 받아와 출력한다. #4에서 읽어야 할 파일을 'example.cfg'로 설정하고, #5에서 해당 설정 파일의 section 정보를 출력하면 정상적으로 값이 출력된다. 이후는 딕셔너리와 같다.

03. 설정 저장

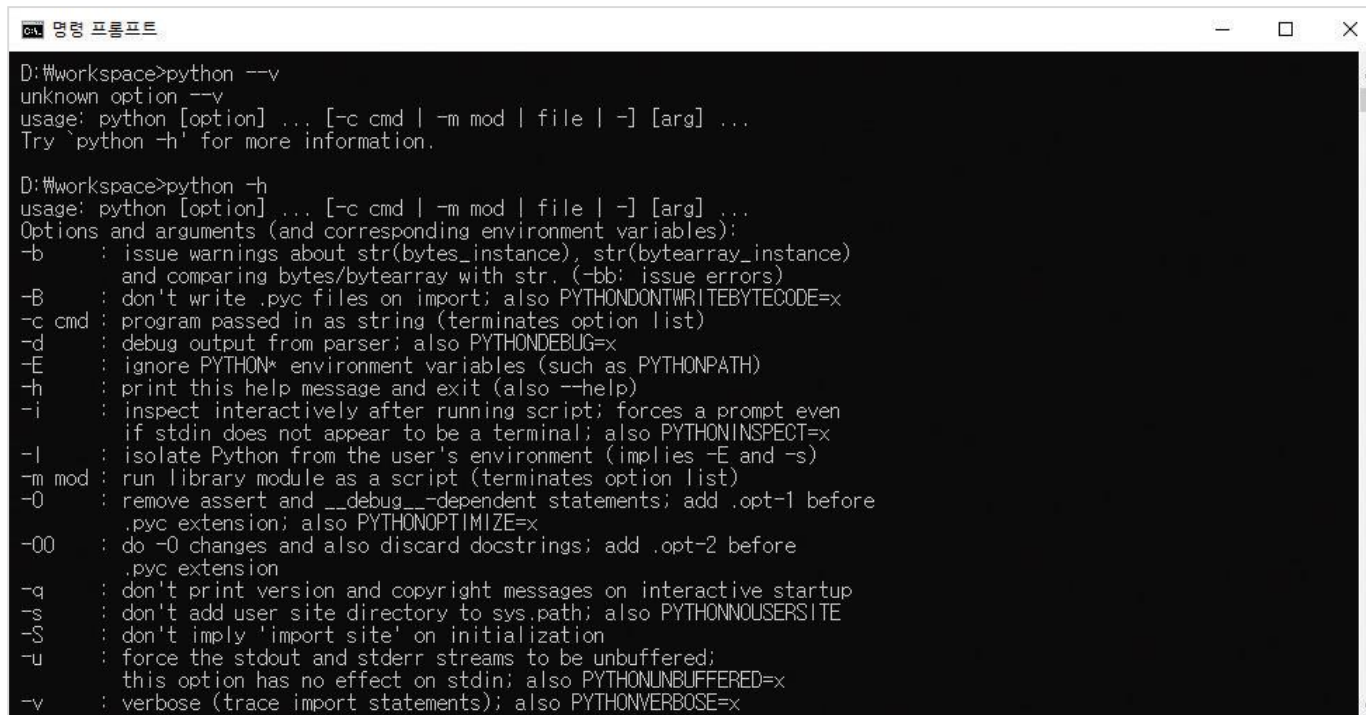
■ argparse 모듈

- argparse 모듈은 configparser 모듈과 달리, 저장된 파일을 사용하는 것이 아니라 프로그램을 콘솔 창에서 실행할 때 세팅을 설정하는 방식이다. 거의 모든 콘솔 프로그램은 실행 시점의 설정 기능을 제공한다.

03. 설정 저장

■ argparse 모듈

- 파이썬을 처음 실행할 때 'python --v'나 'python -h'를 누르면 파이썬 실행과 관련된 도움말을 확인할 수 있다.



```
D:\#workspace>python --v
unknown option --v
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Try `python -h' for more information.

D:\#workspace>python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-b      : issue warnings about str(bytes_instance), str(bytearray_instance)
         and comparing bytes/bytearray with str. (-bb: issue errors)
-B      : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd  : program passed in as string (terminates option list)
-d      : debug output from parser; also PYTHONDEBUG=x
-E      : ignore PYTHON* environment variables (such as PYTHONPATH)
-h      : print this help message and exit (also --help)
-i      : inspect interactively after running script; forces a prompt even
         if stdin does not appear to be a terminal; also PYTHONINSPECT=x
-I      : isolate Python from the user's environment (implies -E and -s)
-m mod  : run library module as a script (terminates option list)
-O      : remove assert and __debug__-dependent statements; add .opt-1 before
         .pyc extension; also PYTHONOPTIMIZE=x
-OO     : do -O changes and also discard docstrings; add .opt-2 before
         .pyc extension
-q      : don't print version and copyright messages on interactive startup
-s      : don't add user site directory to sys.path; also PYTHONNOUSERSITE
-S      : don't imply 'import site' on initialization
-u      : force the stdout and stderr streams to be unbuffered;
         this option has no effect on stdin; also PYTHONUNBUFFERED=x
-v      : verbose (trace import statements); also PYTHONVERBOSE=x
```

[파이썬 커맨드-라인 옵션]

03. 설정 저장

■ argparse 모듈

코드 13-9 arg_sum.py

```
1 import argparse                # argparse 모듈의 호출
2
3 parser = argparse.ArgumentParser(description='Sum two integers.')
4                                # 기본 설정 도움말
5 parser.add_argument('-a', "--a_value", dest="a", help="A integers", type=int)
6                                # a 인수 추가
7 parser.add_argument('-b', "--b_value", dest="b", help="B integers", type=int)
8                                # b 인수 추가
9
10 args = parser.parse_args()     # 입력된 커맨드 라인 인수 파싱
11
12 print(args)                   # 결과 출력
13 print(args.a)
14 print(args.b)
15 print(args.a + args.b)
```

03. 설정 저장

■ argparse 모듈

- ➡ 이 프로그램은 콘솔 창에서 프로그램을 실행할 때, 2개의 명령 행 옵션을 넣고 이를 더하는 결과를 출력하는 프로그램이다.
- ➡ 먼저 1행에서 argparse 모듈을 호출한다. 3행에서는 사용하기 위한 ArgumentParser 객체를 만들고, 기본적인 도움말을 description 변수에 등록한다. 5행과 6행에서는 새로운 인수를 추가한다. 인수를 추가할 때는 -을 이용하여 짧게 쓸 때 사용하는 이름, --을 이용하여 길게 쓸 때 사용하는 이름, 인수의 표시 이름, 도움말, 자료형 등을 함께 등록할 수 있다. 8행에서는 명령을 통해 커맨드 라인에 입력된 인수를 불러와 args 변수에 할당한다. 10~13행에서 각각의 값을 입력하면 실행 결과가 출력된다.

03. 설정 저장

■ argparse 모듈

- [코드 13-9]를 'arg_sum.py'로 저장하고, cmd 창에 'python arg_sum.py -a 5 -b 3'으로 입력하면 다음과 같이 출력된다.

```
Namespace(a=5, b=3)
```

```
5
```

```
3
```

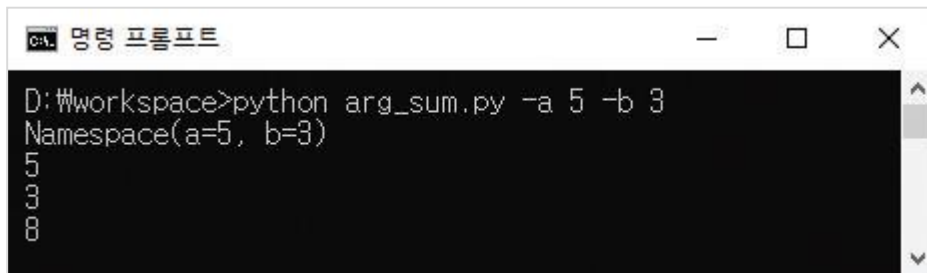
```
8
```

← 값이 저장된 형태는 'Namespace' 객체

← a 인수의 값 출력

← b 인수의 값 출력

← a 인수와 b 인수의 덧셈



```
C:\> 명령 프롬프트
D:\workspace>python arg_sum.py -a 5 -b 3
Namespace(a=5, b=3)
5
3
8
```

03. 설정 저장

■ argparse 모듈

- 만약 커맨드-라인 인수에 대한 정보를 모른다면 어떻게 할 수 있을까? cmd 창에 'pythonarg_sum.py -h'를 입력하여 -h 옵션을 사용하면 해당 값들의 자세한 사용법을 확인할 수 있다.

```
usage: arg_sum.py [-h][-a A][-b B]
```

```
Sum two integers.
```

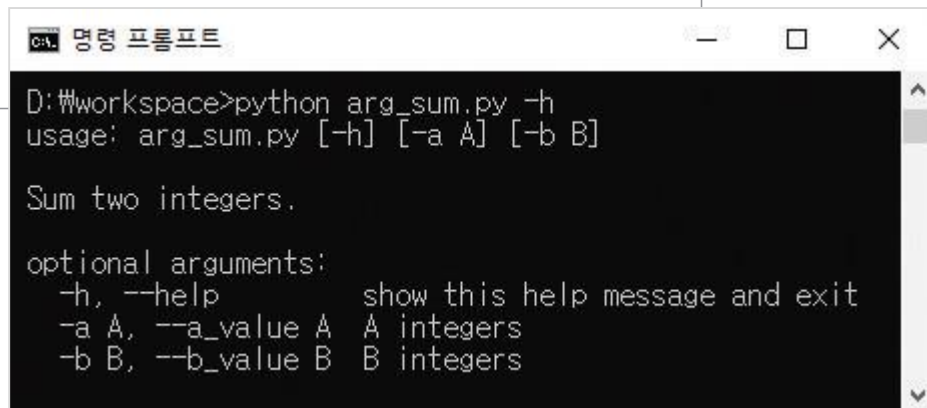
```
optional arguments:
```

```
-h, --help            show this help message and exit
```

```
-a A, --a_value A    A integers
```

```
-b B, --b_value B    B integers
```

[python arg_sum.py -a 5 -b 3'을 입력한 결과]



```
D:\workspace>python arg_sum.py -h
usage: arg_sum.py [-h] [-a A] [-b B]

Sum two integers.

optional arguments:
  -h, --help            show this help message and exit
  -a A, --a_value A    A integers
  -b B, --b_value B    B integers
```


04

Lab: 로깅 프로그램

04. Lab: 로깅 프로그램

■ logging formatter

- formatter는 로그의 결과 값을 일정 형식을 지정하여 출력하는 기능이다

코드 13-10 loggingformatter.py

```
1 import logging
2
3 logger = logging.getLogger('myapp')           # Logger 생성
4 hdlr = logging.FileHandler('myapp.log')       # FileHandler 생성
5
6 formatter = logging.Formatter('%(asctime)s %(levelname)s %(process)d
7                                %(message)s')   # Logging Formatter 생성: 시간, 로깅 레벨, 프로세스 ID, 메시지
8
9 hdlr.setFormatter(formatter)                  # FileHandler에 formatter 등록
10 logger.addHandler(hdlr)                      # Logger에 'FileHandler' 등록
11
12 logger.setLevel(logging.INFO)                 # 로깅 레벨 설정
13
14 logger.error('ERROR occurred')                # 로깅 정보 출력
15 logger.info('HERE WE ARE')
16 logger.info('TEST finished')
```

04. Lab: 로깅 프로그램

■ logging formatter

- ➔ 먼저 3행에서 Logger를 생성하고, 4행에서 출력을 결정하는 Handler를 생성한다. 6행에서 Formatter 객체를 생성하면서 메시지 출력 방식을 결정한다. [코드 13-10]에서는 시간(asctime), 로깅 레벨(levelname), 프로세스(process ID), 메시지(message) 등이 출력되도록 설정하였고, 각 변수 뒤 s는 문자열, d는 정수형을 뜻한다. 8 · 9행에서는 생성된 객체를 Logger 객체에 등록한다. 먼저 Handler 객체에 Formatter 객체를 등록하고, 다음으로 Logger 객체에서 Handler를 등록한다. 마지막으로 10행에서 로깅 레벨을 설정한다. 여기서 코드를 실행하면, FileHandler를 사용해 'myapp.log'라는 파일에 기록하였기 때문에 'myapp.log' 파일이 생성되고 다음 내용이 들어 있을 것이다.

```
2018-11-05 15:26:51,801 ERROR 8112 ERROR occurred
2018-11-05 15:26:51,801 INFO 8112 HERE WE ARE
2018-11-05 15:26:51,801 INFO 8112 TEST finished
```

04. Lab: 로깅 프로그램

■ logging formatter

- logging config file은 파일에 로깅과 관련된 정보를 저장하여, 실제 사용할 때 파일을 생성하는 것만으로 로깅을 설정하는 방법이다. 긴 코드를 쓰지 않고 매우 간단하게 로깅을 설정할 수 있는 장점이 있다.

```
[loggers]
keys=root

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler
```

04. Lab: 로깅 프로그램

■ logging formatter

```
[handler_consoleHandler]
class=StreamHandler
level==DEBUG
formatter=simpleFormatter
args=(sys.stdout,)

[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
datefmt=
```

- ➡ 로깅 설정 파일의 구조는 하나의 section을 만들고, _를 사용해 그 section에서 사용하는 상세한 설정을 구축하는 방식이다. 코드를 만지는 것보다 설정 파일을 만지는 것이 사용자 입장에서 훨씬 부담이 덜하고 사용하기 쉽다는 장점이 있다.

04. Lab: 로깅 프로그램

■ logging formatter

➡ 실제 설정 파일을 사용하기 위해서는 다음과 같이 설정 파일을 호출한다.

```
logging.config.fileConfig('logging.conf')  
logger = logging.getLogger()
```

04. Lab: 로깅 프로그램

■ 실습

코드 13-11 loggingprogram.py

```
1 import logging
2 import logging.config
3 import csv
4
5 # 모듈 호출
6 logging.config.fileConfig('logging.conf')           # Logger 생성
7 logger = logging.getLogger()
8
9 line_counter = 0
10 data_header = []
11 employee = []
12 customer_USA_only_list = []
13 customer = None
14
15 # 변수 선언 등 생략
16 logger.info('Open file {0}'.format("TEST",))
```

04. Lab: 로깅 프로그램

■ 실습

```
17 try:
18     with open("customers.csv", "r") as customer_data:
19         customer_reader = csv.reader(customer_data, delimiter=',', quotechar='')
20         for customer in customer_reader:
21             if customer[10].upper()=="USA": # customer 데이터의 offset 10번째 값
22                 logger.info('ID {0} added'.format(customer[0],))
23                 customer_USA_only_list.append(customer) # country 필드가 "USA"
                                                           인 것만
24 except FileNotFoundError as e:
25     logger.error('File NOT found {0}'.format(e,))
26
27 logger.info('Write USA only data at {0}'.format("customers_USA_only.csv",))
28 with open("customers_USA_only.csv", "w") as customer_USA_only_csv:
29     for customer in customer_USA_only_list:
30         customer_USA_only_csv.write(",".join(customer).strip('\n')+"\n")
31
32 logger.info('Program finished')
```


04. Lab: 로깅 프로그램

■ 실습

```
2018-11-15 11:40:13,184 - root - INFO - Open file TEST
2018-11-15 11:40:13,184 - root - INFO - ID 112 added
2018-11-15 11:40:13,184 - root - INFO - ID 124 added
2018-11-15 11:40:13,184 - root - INFO - ID 129 added
2018-11-15 11:40:13,184 - root - INFO - ID 131 added
: ← 생략
2018-11-15 11:40:13,186 - root - INFO - ID 487 added
2018-11-15 11:40:13,186 - root - INFO - ID 495 added
2018-11-15 11:40:13,186 - root - INFO - Write USA only data at customers_USA_
only.csv
2018-11-15 11:40:13,188 - root - INFO - Program finished
```

Thank You !