

따라하며 배우는

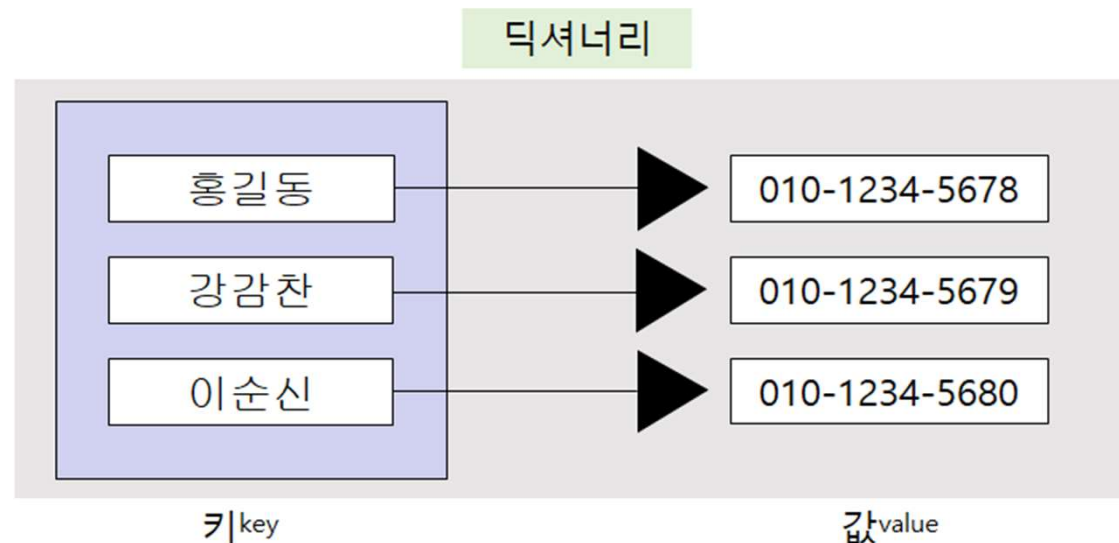
# 파이썬과 데이터 과학



8장 연관된 데이터를  
딕셔너리로 짤 짓자

## 8.1 키와 값을 가진 딕셔너리로 자료를 저장하자

- **딕셔너리** dictionary도 리스트와 같이 값을 저장하는 자료구조로 파이썬에서는 기본 자료형으로 제공되고 있다.
- 하지만 딕셔너리에는 **값** value과 관련된 **키** key가 있다는 것이 큰 차이점이다.

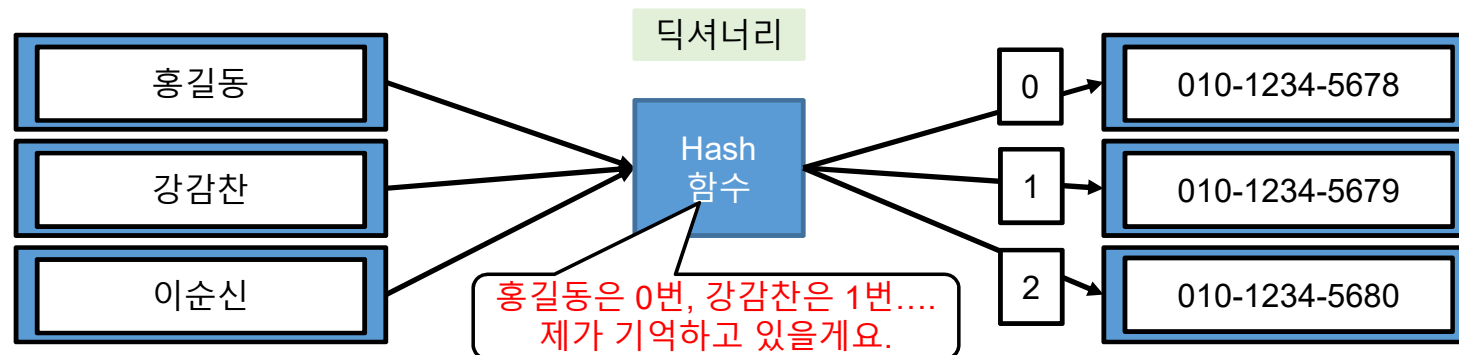


## 8.1 키와 값을 가진 딕셔너리로 자료를 저장하자

- 리스트와 딕셔너리의 차이점
- 리스트는 순차적인 배열로 구현되어 있음

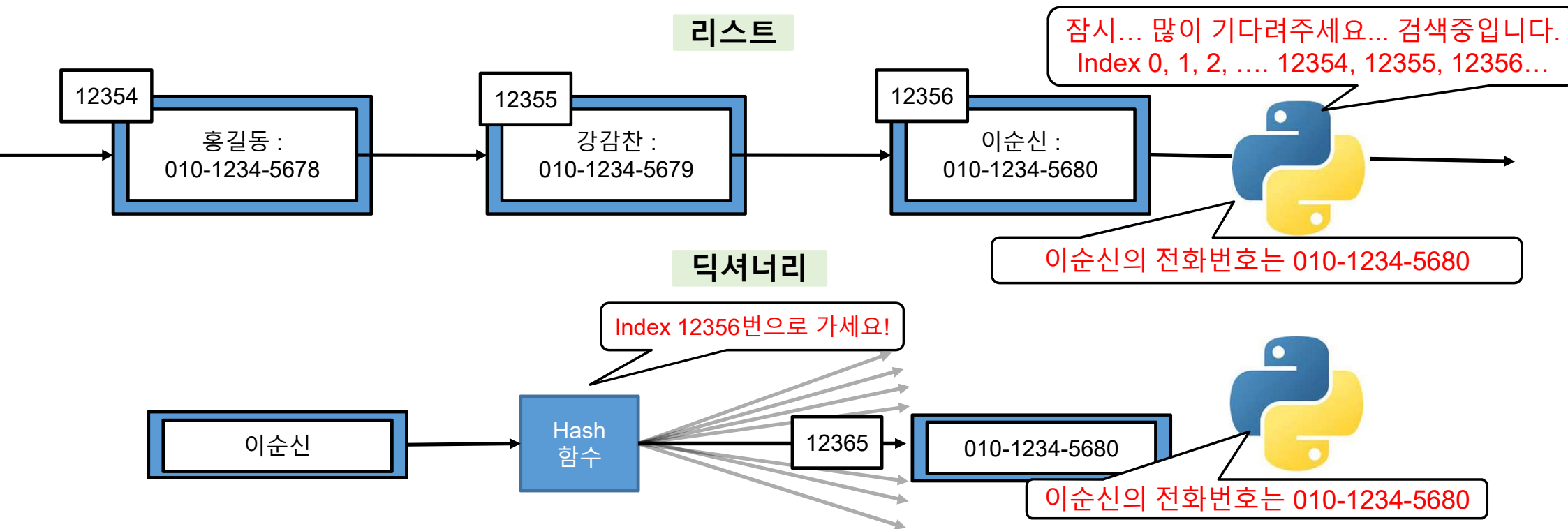


- 딕셔너리는 key를 입력으로 받아 value가 저장된 인덱스를 반환하는 함수인 **Hash 함수**를 통해 구현되어 있음



## 8.2 딕셔너리의 기능을 알아보자

- 딕셔너리에서 가장 중요한 연산은 키를 가지고 연관된 값을 찾는 것
- 주소록에서 “**이순신**” 이름을 가지고 전화번호를 찾는 일을 생각해보자.  
리스트에서는 이름을 검색해서 항목을 찾아내야 하지만, 딕셔너리에서는 Hash 함수로 구현되어 있기 때문에 키만 가지고 즉시 찾을 수 있다.



## 8.1 키와 값을 가진 딕셔너리로 자료를 저장하자

- 파이썬의 딕셔너리에서는 서로 관련되어 있는 키와 값도 함께 저장되는데, 이것을 **키-값 쌍** *key-value pair*이라고 한다.
- 딕셔너리를 만드는데는 몇 가지 방법이 있지만 일단 `{ }`를 이용해서 공백 딕셔너리를 생성하고 여기에 하나씩 전화번호를 추가해보자.
- 공백 리스트는 대괄호 `[ ]`로 생성하고, **딕셔너리는 중괄호 `{ }`로 생성**한다는 것에 유의하자.

```
>>> phone_book = { }    # 공백 딕셔너리를 생성
```

```
>>> phone_book["홍길동"] = "010-1234-5678"
```

```
>>> print(phone_book)
{'홍길동': '010-1234-5678'}
```

## 8.1 키와 값을 가진 딕셔너리로 자료를 저장하자

- 딕셔너리를 생성하면서 동시에 초기화하는 방법도 있다.

```
>>> phone_book = {"홍길동": "010-1234-5678"}
```

- phone\_book 딕셔너리를 출력하면 딕셔너리의 항목item이  
    쉼표로 구분되어 출력된다.
- 이제 이 딕셔너리에 몇 개의 다른 전화번호를 추가해서  
    출력해보면 다음과 같다.

```
>>> phone_book["강감찬"] = "010-1234-5679"  
>>> phone_book["이순신"] = "010-1234-5680"  
>>> print(phone_book)  
{'이순신': '010-1234-5680', '홍길동': '010-1234-5678', '강감찬': '010-1234-5679'}
```

## 8.2 딕셔너리의 기능을 알아보자

- 딕셔너리에서 사용되는 모든 키를 출력하려면 다음과 같이 `keys()`라는 메소드를 사용한다.

```
>>> phone_book.keys()  
dict_keys(['이순신', '홍길동', '강감찬'])
```

- 반면 딕셔너리에서 사용되는 모든 값을 출력하려면 `values()`를 사용한다.

```
>>> phone_book.values()  
dict_values(['010-1234-5680', '010-1234-5678', '010-1234-5679'])
```

- 그리고, 딕셔너리 내부의 모든 값을 출력하려면 `items()`를 사용할 수도 있다.
- for문에서 `phone_book.item()`와 같은 함수를 호출하면 (키, 값) 튜플이 반환된다.

```
>>> phone_book.items()  
dict_items([('홍길동', '010-1234-5678'), ('강감찬', '010-1234-5679'), ('이순신', '010-1234-5680')])  
>>> for name, phone_num in phone_book.items():  
...     print(name,':', phone_num)  
...  
홍길동 : 010-1234-5678  
강감찬 : 010-1234-5679  
이순신 : 010-1234-5680
```

딕셔너리의 항목들을  
시퀀스로 추출할 수 있다.

## 8.3 딕셔너리의 다양하고 멋진 기능들을 수행하는 메소드

- 딕셔너리의 모든 항목을 하나씩 출력하려면 리스트처럼 for 루프를 사용하자.
- `keys()` 메소드는 딕셔너리에 있는 키 항목을 시퀀스로 반환하므로, 이 값을 받아서 `phone_book[key]`로 접근해보자.

```
>>> for key in phone_book.keys():  
...     print(key, ': ', phone_book[key])  
...  
홍길동 : 010-1234-5678  
강감찬 : 010-1234-5679  
이순신 : 010-1234-5680
```

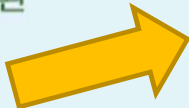
- 딕셔너리 안에서 항목들은 자동으로 정렬되지 않는다.
- 그래서 우리는 `sorted()` 함수를 사용하여 **딕셔너리의 키를 기준으로** 정렬을 수행할 수 있다.



## 8.3 딕셔너리의 다양하고 멋진 기능들을 수행하는 메소드

- sorted() 함수는 phone\_book.item()와 같이 키, 값의 튜플 쌍을 받아 이를 정렬하여 리스트로 반환한다.
- 잠시 뒤에 배울 **람다 표현**은을 통해 x를 인자로 받아 x의 첫 항목인 x[0]를 반환하는 기능을 한다.

```
>>> sorted(phone_book)           # 딕셔너리를 키를 기준으로 정렬하며 리스트를 반환
['강감찬', '이순신', '홍길동']
>>> sorted_phone_book = sorted(phone_book.items(), key=lambda x: x[0])
>>> print(sorted_phone_book)
[('강감찬', '010-1234-5679'), ('이순신', '010-1234-5680'), ('홍길동', '010-1234-5678')]
```



X[0]	X[1]
이순신	010-1234-5680

- 만일 딕셔너리의 항목을 삭제하려면 다음과 같이 del을 사용한다.

```
>>> del phone_book["홍길동"]      # "홍길동" 키를 이용하여 딕셔너리의 한 항목 삭제
>>> print(phone_book)
{'강감찬': '010-1234-5679', '이순신': '010-1234-5680'}
```

## 8.3 딕셔너리의 다양하고 멋진 기능들을 수행하는 메소드

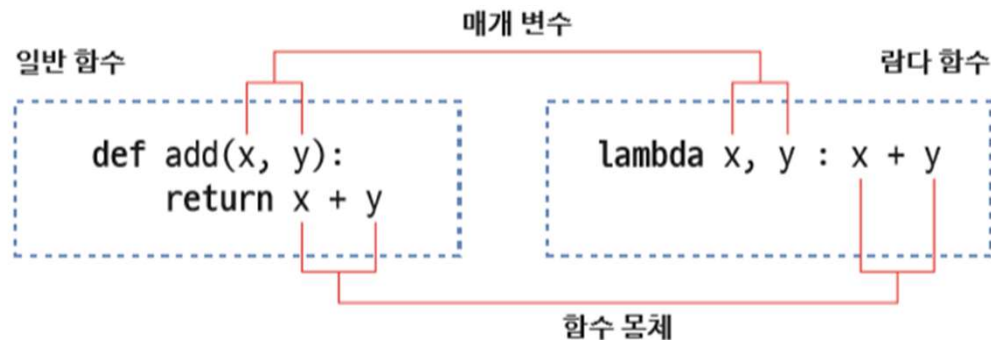
메소드	하는 일
<code>keys()</code>	딕셔너리 내의 모든 키를 반환한다.
<code>values()</code>	딕셔너리 내의 모든 값을 반환한다.
<code>items()</code>	딕셔너리 내의 모든 항목을 [키]:[값] 쌍으로 반환한다.
<code>get(key)</code>	키에 대한 값을 반환한다. 키가 없으면 <code>None</code> 을 반환한다.
<code>pop(key)</code>	키에 대한 값을 반환하고, 그 항목을 삭제한다. 키가 없으면 <code>KeyError</code> 예외를 발생시킨다.
<code>popitem()</code>	제일 마지막에 입력된 항목을 반환하고 그 항목을 삭제한다.
<code>clear()</code>	딕셔너리 내의 모든 항목을 삭제한다.

## 8.4 람다 함수 = 이름이 없는 함수

- 람다 함수란 이름이 없는 함수로 정의할 수 있는데 간단한 1회용 작업에 유용하다.
- 가끔씩은 함수를 만들지 않고 함수화된 기능만을 불러 사용하고자 할 경우가 있기에 **람다 표현식** lambda expression이라고도 한다.
- 람다 함수의 사용시 주의할 점은 표현식 안에서 새로운 변수를 선언할 수 없다는 것이다.
- 그리고 **람다 함수의 반환 값은 변수 없이 식 한 줄로 표현할 수 있어야 하기 때문에 복잡한 기능의 함수가 필요하다면 def 키워드로 함수를 정의하여야 한다.**

## 8.4 람다 함수 = 이름이 없는 함수

- 이제 두 값을 인자로 받아서 그 합을 반환하는 일반 함수와 람다함수의 차이점을 아래 그림으로 비교해보자.



- 두 수를 입력받아 그 합을 반환하는 익명의 함수 `lambda x, y : x+y`는 다음과 같이 호출할 수도 있다.

```
>>> print('100과 200의 합 :', (lambda x, y: x + y)(100, 200)) # 100, 200이 람다 함수의 인자
100과 200의 합 : 300
```

## 8.4 람다 함수 = 이름이 없는 함수

- 만일 특정한 튜플에서 첫 항목만을 추출하는 람다 함수를 정의하려면 다음과 같이 할 수 있다.

람다 함수로 인자  
x의 x[0] 항목  
추출이  
가능함(여기서는  
인자가 t이므로  
t[0]항목 추출

(100, 200, 300)

t[0] t[1] t[2]

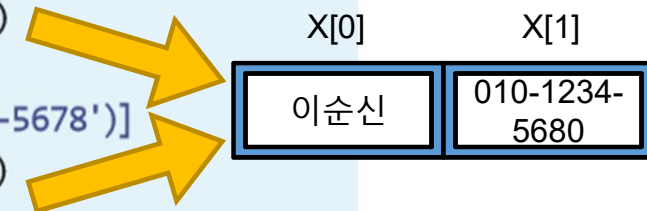
```
>>> t = (100, 200, 300)
>>> (lambda x: x[0])(t) # t를 인자로 받아서 그 첫 항목 t[0]을 반환한다
100
>>> (lambda x: x[1])(t) # t를 인자로 받아서 그 두 번째 항목 t[1]을 반환한다
200
```

- 위의 `lambda x: x[0]` 표현식은 임의의 항목을 가진 객체에 대하여 그 첫 번째 항목을 반환하는 기능을 한다.

## 8.4 람다 함수 = 이름이 없는 함수

- 만일 `x[1]`을 반환하도록 하면 아래와 같이 두 번째 항목인 전화번호를 가지고 정렬을 수행하게 된다.

```
>>> print(phone_book.items()) # 딕셔너리의 items()는 키, 값을 튜플로 출력
dict_items([('홍길동', '010-1234-5678'), ('강감찬', '010-1234-5679'), ('이순신', '010-1234-5680')])
>>> # 항목의 첫 인자인 이름을 기준으로 정렬한다 : 한글 사전 순서
>>> sorted_phone_book1 = sorted(phone_book.items(), key=lambda x: x[0])
>>> print(sorted_phone_book1)
[('강감찬', '010-1234-5679'), ('이순신', '010-1234-5680'), ('홍길동', '010-1234-5678')]
>>> sorted_phone_book2 = sorted(phone_book.items(), key=lambda x: x[1])
>>> print(sorted_phone_book2)
[('홍길동', '010-1234-5678'), ('강감찬', '010-1234-5679'), ('이순신', '010-1234-5680')]
```





## 도전문제 8.2

위의 프로그램을 편의점의 재고를 관리하는 프로그램으로 업그레이드해보자. 즉 재고를 증가, 또는 감소시킬 수도 있도록 코드를 추가하여 보자. 재고조회, 입고, 출고와 같은 간단한 메뉴도 만들어보자.

메뉴를 선택하시오 1) 재고조회 2) 입고 3) 출고 4) 종료 : 1

[재고조회] 물건의 이름을 입력하시오: 콜라

재고 : 4

메뉴를 선택하시오 1) 재고조회 2) 입고 3) 출고 4) 종료 : 2

[입고] 물건의 이름과 수량을 입력하시오 : 콜라 4

콜라의 재고 : 8

메뉴를 선택하시오 1) 재고조회 2) 입고 3) 출고 4) 종료 : 4

프로그램을 종료합니다.





## 도전문제 8.2

위의 프로그램을 편의점의 재고를 관리하는 프로그램으로 업그레이드해보자. 즉 재고를 증가, 또는 감소시킬 수도 있도록 코드를 추가하여 보자. 재고조회, 입고, 출고와 같은 간단한 메뉴도 만들어보자.

```
메뉴를 선택하시오 1) 재고조회 2) 입고 3) 출고 4) 종료 : 1
[재고조회] 물건의 이름을 입력하시오: 콜라
재고 : 4
메뉴를 선택하시오 1) 재고조회 2) 입고 3) 출고 4) 종료 : 2
[입고] 물건의 이름과 수량을 입력하시오 : 콜라 4
콜라의 재고 : 8
메뉴를 선택하시오 1) 재고조회 2) 입고 3) 출고 4) 종료 : 4
프로그램을 종료합니다.
```

```
stock_dict = {} # 빈 딕셔너리 선언

while True: # 종료시 까지 계속 반복
    menu = int(input("메뉴를 선택하시오 1) 재고조회 2) 입고 3) 출고 4) 종료 :"))

    if menu == 1: # 재고조회
        name = input("[재고조회] 물건의 이름을 입력하시오: ")
        print("재고 :", stock_dict[name]) # 이름을 key로 재고 조회

    elif menu == 2: # 입고
        name, num = (input("[입고] 물건의 이름과 수량을 입력하시오 :")).split()
        num = int(num)
        if name in stock_dict.keys(): # 이름이 재고 딕셔너리 있으면
            stock_dict[name] += num # 재고를 num 만큼 증가
        else:
            stock_dict[name] = num # 없으면 key를 name, value를 num으로 할당
        print("재고 :", stock_dict[name]) # 이름을 key로 재고 조회

    elif menu == 3:
        raise Exception("출고는 구현되지 않았습니다.")

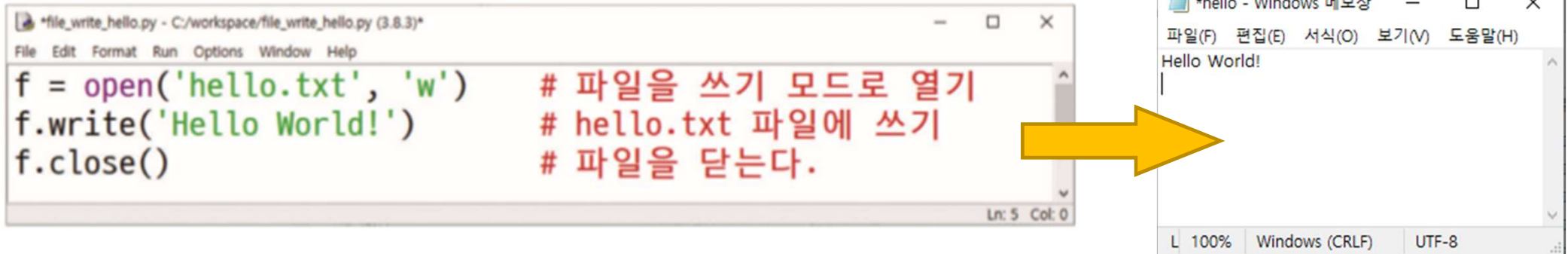
    elif menu == 4: # 종료 (break)
        break
```



## 8.9 파일로부터 자료를 읽고 저장해보자

- 컴퓨터 **파일**file이란 컴퓨터의 저장 장치 내에 데이터를 저장하기 위해 사용하는 논리적인 단위를 말한다.
- 파일은 하드 **디스크**hard disk나 **외장 디스크**external disk 같은 저장 장치에 저장한 후 필요할 때 다시 불러서 사용하는 것이 가능하며, 필요에 따라서 수정하는 것도 가능하다.
- 파일에는 여러 종류가 있으며 일반적으로 마침표(.)문자 뒤에 py, txt, doc, hwp, pdf와 같은 확장자를 붙여서 파일의 종류를 구분한다.

## 8.9 파일로부터 자료를 읽고 저장해보자



- 이 코드는 **open()** 이라는 명령을 통해서 'hello.txt' 파일을 열게 되는데 뒤에 나타나는 'w' 인자에 의해서 파일을 쓰기 모드로 열게 된다.
- 이렇게 만든 파일 객체 `f` 는 **f.write()** 명령을 통해서 'hello world!' 라는 문자열을 현재 디렉토리의 `hello.txt` 라는 파일에 쓰고 모든 작업을 마친 후 **f.close()** 를 통해서 파일 쓰기 작업을 종료한다.
- 파일 작업은 운영체제에 접근권한을 요청해야 하는 시스템 자원이다. **사용한 후엔 꼭 close() 함수로 파일 권한을 반납**해야 한다.

## 8.9 파일로부터 자료를 읽고 저장해보자

- 역시 `open()` 함수를 사용하여야 하지만 'w' 인자가 아닌 'r' 인자를 **사용해야 읽기 모드(read mode)**로 파일을 읽을 수 있다.
- 성공적으로 파일 읽기가 완료되면 다음과 같이 파일의 내용을 화면에서 볼 수 있다.

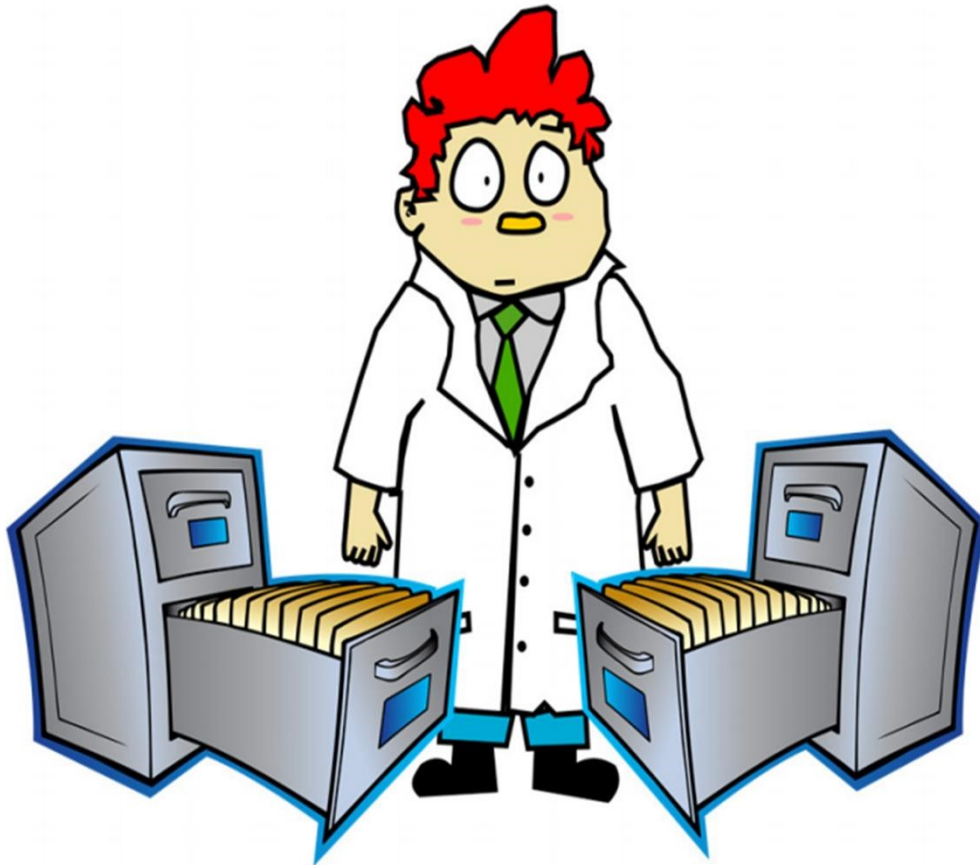
```
f = open('hello.txt', 'r')    # 파일을 연다.  
s = f.read()                 # hello.txt 파일을 읽는다.  
print(s)                     # 파일의 내용을 출력한다.  
f.close()                    # 파일을 닫는다.
```

```
Hello World!
```

- 파일에 여러줄의 내용이 있을 경우 `for`문을 사용할 수 있다.

## PART 2

# 데이터 과학과 인공지능



## 9장 텍스트를 처리해보자

따라하며 배우는

# 파이썬과 데이터 과학

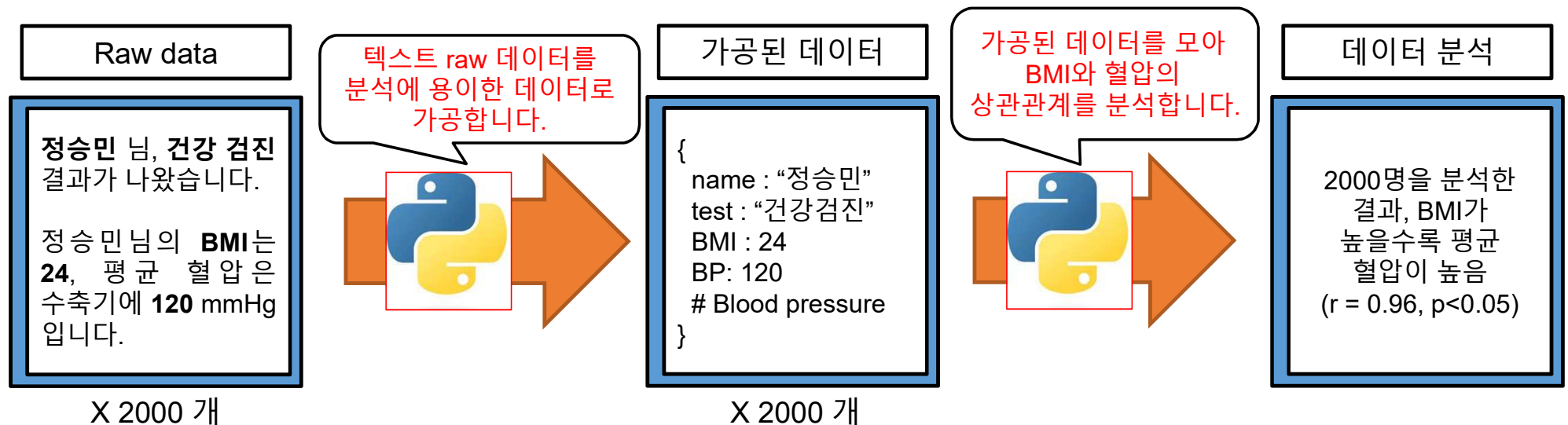


## 책과 다른점

- 우리의 교재에서는 워드클라우드, 정규표현식 등 텍스트 처리에 대해 조금 더 깊이 있는 주제를 다룬다.
- 특히, **정규표현식(Regular Expression)**은 텍스트를 처리 하는데에 매우 강력한 기능이므로 텍스트 데이터를 많이 다루게 되는 웹 프로그래머가 되려면 꼭 알아야하는 기능이다.
- 하지만, 본 강의에서는 텍스트 데이터 (트윗, 카카오톡, 상품 리뷰 등)의 해석이 아닌 **수치 데이터** (생체 신호, 의료 데이터, 주식의 가격 등)을 해석하는데에 집중하기 위해서 이에 필요한 기본적인 텍스트 처리만을 다룬다.
- 이는 **현실의 대부분 현상은 수치로 나타낼 수 있고 이를 분석**하는 것이 일반적이고 중요한 프로그래밍이기 때문이다. 특수한 경우인 텍스트 데이터 분석 역시 중요하지만 본 강의에서는 수치데이터에 집중한다.

## 9.1 텍스트 데이터란 무엇인가

- 텍스트 데이터는 구조화된 문서(HTML, XML, CSV, JSON 파일)와 구조화되지 않은 문서(자연어로 된 텍스트)로 나눌 수 있다.
- 일반적으로 **원천(raw) 데이터는 가공된 형태가** 아니기 때문에 우리는 이들 데이터를 수정하여서 완전한 데이터로 만들어야 한다.



## 9.2 문자열에서 개별 문자들을 뽑아보자

- 문자열에서 사용할 수 있는 가장 기본적인 작업은 아마도 문자열 안에 있는 개별 문자들을 추출하는 작업일 것이다

						[6:10]					
0	1	2	3	4	5	6	7	8	9	10	11
M	o	n	t	y		P	y	t	h	o	n
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
[-12:-7]											

예를 들어 'Monty Python'이라는 문자열이 변수 `s`에 저장되어 있다고 하자.

```
>>> s = 'Monty Python'
>>> s[0]
'M'
```

- 위의 예제와 같이 `s[0]`과 같이 인덱스 `0`을 이용해서 문자 'M'에 접근 가능하며, `s[11]`은 마지막 문자 'n'이 된다.

## 9.2 문자열에서 개별 문자들을 뽑아보자

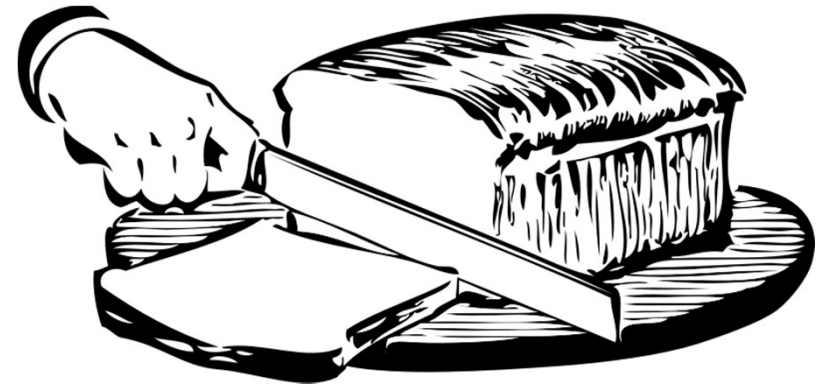
```
>>> s[6:10]
'Pyth'
>>> s[-12:-7]
'Monty'
```

- 문자열의 오른쪽 끝의 두 문자만을 제외하고 슬라이싱하여 복사하려 하면 다음과 같은 표현식을 사용할 수 있다.

```
>>> t = s[:-2]
>>> t
'Monty Pyth'
```

- 만일 오른쪽 끝에 있는 두 개의 문자를 선택하려면 `s[-2:]`를 사용하면 된다. 그러므로 `s[:-2] + s[-2:]`는 원래 문자열이 된다.

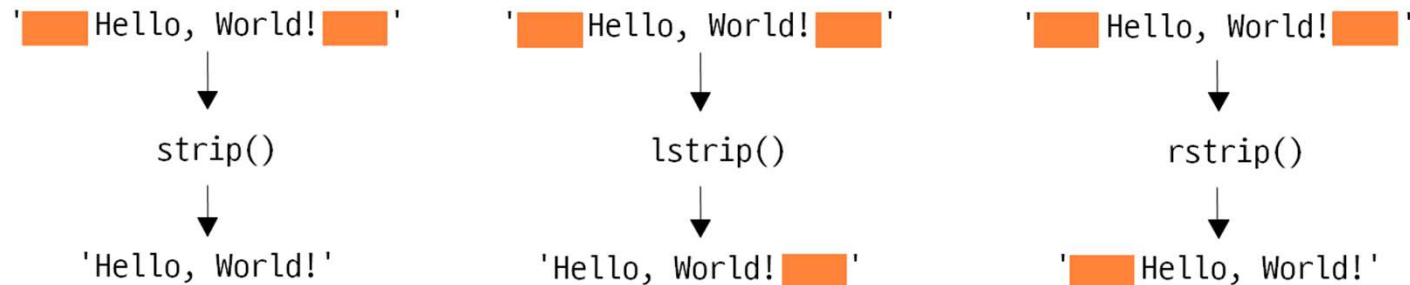
```
>>> t = s[-2:]          # t는 s의 인덱스 -2에서부터 마지막 원소까지 가져온다
>>> t
'on'
>>> s[:-2] + s[-2:]
'Monty Python'
```





## 9.5 대문자와 소문자 변환, 그리고 문자열 삭제4

- 문자열 데이터를 처리할 때 문자열에서 원치 않는 공백을 제거하는 작업은 **strip()** 함수가 담당한다.



```
>>> s = ' Hello, World! '
>>> s.strip()          # 왼쪽과 오른쪽의 공백문자를 모두 제거한다
'Hello, World!'
>>> s.lstrip()         # 왼쪽의 공백문자만 제거한다
'Hello, World! '
>>> s.rstrip()        # 오른쪽의 공백문자만 제거한다
' Hello, World!'
```

## 9.5 대문자와 소문자 변환, 그리고 문자열 삭제

- 만일 문자열의 앞과 뒤에있는 특정한 문자를 삭제하려면 문자를 `strip()`의 인자로 이 문자를 전달한다.

```
>>> s = '#####this is an example#####'
>>> s.strip('#')          # 문자열의 앞 뒤에 있는 해시문자를 제거한다
'this is an example'
```

- `rstrip()`과 `lstrip()`에도 특정한 문자를 인자로 넣어줄 수 있다.

```
>>> s = '#####this is an example#####'
>>> s.lstrip('#')
'this is an example#####'
>>> s.rstrip('#')
'#####this is an example'
>>> s.strip('#').capitalize() # 삽문자를 제거하고 문장의 첫글자를 대문자로 만든다
'This is an example'
```

## 9.3 split() 메소드는 문자열을 잘 잘라줘요

- 문자열 안의 단어들이 쉼표나 빈칸 등의 구분자로 분리되어 있다고 하자.
- 예를 들어 다음과 같은 `s` 문자열에 **`split()`**이라는 메소드를 사용하면 공백을 구분자로 사용하여 하나의 문자열을 3개의 문자열로 분리할 수 있다.

```
>>> s = 'Welcome to Python'
>>> s.split()
['Welcome', 'to', 'Python']
```

- 반면 다음과 같이 마침표로 구분된 연,월,일이 있을 경우 마침표(.)를 **`split()`** 메소드의 인자로 주면 마침표 단위로 문자를 구분할 수 있다.

```
>>> s = '2021.8.15'
>>> s.split('.')
['2021', '8', '15']
```

## 9.3 split() 메소드는 문자열을 잘 잘라줘요

- 그러나 이 방법은 공백이 두 칸 이상이거나 앞뒤에 공백이 있을 경우에 적용할 수 없으므로 아래와 같이 **strip()**을 이용하여 공백을 제거하는 것이 바람직할 것이다.

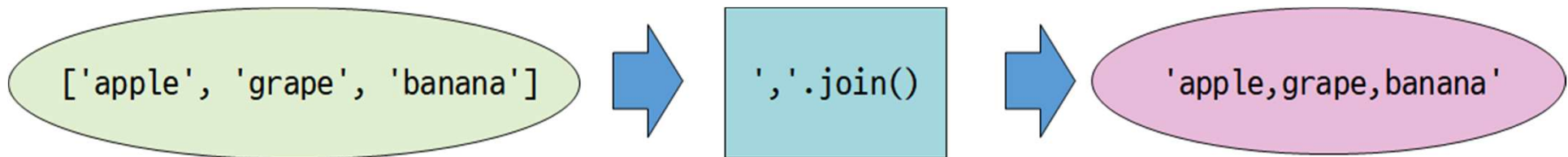
```
>>> s = 'Welcome, to, Python, and , bla, bla '
>>> [x.strip() for x in s.split(',')]  
['Welcome', 'to', 'Python', 'and', 'bla', 'bla']
```

- 만일 문자열을 모두 개별 문자들로 분해하려면 어떻게 하면 될까? **list()**를 호출해주면 된다.

```
>>> list('Hello, World!')  
['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
```

## 9.4 문자열을 이어붙이는 것은 파이썬한테는 쉬운 일

- **split()**가 문자열을 부분 문자열들로 분리하는 함수라면 **join()**은 반대로 부분 문자열들을 모아서 하나의 문자열로 만드는 역할을 하는 함수이다. **join()**을 호출할 때는 접착제 역할을 하는 separator 문자를 지정할 수 있다.



```
>>> ','.join(['apple', 'grape', 'banana'])  
'apple,grape,banana'
```

쉼표를 이용하여 세 단어를  
연결함.

## 9.4 문자열을 이어붙이는 것은 파이썬한테는 쉬운 일

- 아래의 예제에서 보면 `join()`은 separator 문자를 문자열 사이에만 넣고 문자열의 앞이나 뒤에는 넣지 않는 것을 알 수 있다.

```
>>> '-'.join('010.1234.5678'.split('.')) # .으로 구분된 전화번호를 하이픈으로 고치기
'010-1234-5678'
```

- 위 예제는 문자를 다른 문자로 대체하는 `replace()` 함수를 통해서도 똑같이 할 수 있다.

```
>>> '010.1234.5678'.replace('.', '-')
'010-1234-5678'
```

- 또한 다음과 같이 `list()` 함수로 분리한 문자들을 모아서 다시 원래의 문자열로 만들때도 `join()`을 사용한다.

```
>>> s = 'hello world'
>>> clist = list(s)
>>> clist
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> ''.join(clist)
'hello world'
```

## 9.5 대문자와 소문자 변환, 그리고 문자열 삭제

- 문자열에서 대문자를 소문자로 변경하는 함수는 **lower()**이고 반대는 대문자로 변경하는 메소드는 **upper()** 함수이다. 첫 번째 문자만 대문자로 변환하는 함수는 **capitalize()**이다.

```
>>> s = 'Hello, World!'
>>> s.lower()
'hello, world!'
>>> s.upper()
'HELLO, WORLD!'
```

- 문자열에서 **count()** 메소드는 문자열 중에서 부분 문자열이 등장하는 횟수를 반환한다.

```
>>> s = 'www.booksr.co.kr'      # 생능출판사의 홈페이지
>>> s.count('.')                # . 이 몇번 나타나는가를 알려준다
3
```

- 위에서 언급한 메소드 말고 파이썬 내장함수도 텍스트 데이터에 적용할 수 있다. **len()** 함수는 문자열의 길이를 반환한다.

## 9.5 대문자와 소문자 변환, 그리고 문자열 삭제

- **find()** 메소드는 문자열에서 지정된 부분 문자열을 찾아서 그 인덱스를 반환한다. **지정된 문자를 찾지 못했을 경우에는 -1을 반환한다.** 문자열 중에서 관심 있는 부분을 찾을 때 **find()** 함수를 사용하면 좋다.

```
>>> s = 'www.booksr.co.kr'
>>> s.find('.kr')
13
>>> s.find('x')      # 'x' 문자열이 없을 경우 -1을 반환함
-1
```





## 도전문제 8.2

승민이는 취업을 위해 자소서를 넣고 떨어지기를 반복하다가 써놓은 자소서를 자동으로 다른 기업에 넣을 자소서로 바꿀 수 있는 프로그램을 만들기로 했다. **삼성전자 자소서.txt** 파일을 불러와 “삼성전자”를 “세동전자”로 바꾸어 **세동전자 자소서.txt**로 저장하는 프로그램을 만들어 보자

저는 **삼성전자**에 꼭 들어가고 싶습니다. 제가 얼마나 열심히 공부했는지 아십니까?  
**삼성전자**에 들어가기만 하면 뼈를 묻겠습니다. **삼성전자**에서 뽑아주실때까지 숨 참겠습니다.



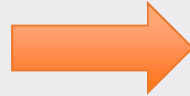
저는 **세동전자**에 꼭 들어가고 싶습니다. 제가 얼마나 열심히 공부했는지 아십니까?  
**세동전자**에 들어가기만 하면 뼈를 묻겠습니다. **세동전자**에서 뽑아주실때까지 숨 참겠습니다.



## 도전문제 8.2

승민이는 취업을 위해 자소서를 넣고 떨어지기를 반복하다가 써놓은 자소서를 자동으로 다른 기업에 넣을 자소서로 바꿀 수 있는 프로그램을 만들기로 했다. **삼성전자 자소서.txt** 파일을 불러와 “삼성전자”를 “세동전자”로 바꾸어 **세동전자 자소서.txt**로 바꾸는 프로그램을 만들어 보자

저는 **삼성전자**에 꼭 들어가고 싶습니다. 제가 얼마나 열심히 공부했는지 아십니까?  
**삼성전자**에 들어가기만 하면 뼈를 문겠습니다.



저는 **세동전자**에 꼭 들어가고 싶습니다. 제가 얼마나 열심히 공부했는지 아십니까?  
**세동전자**에 들어가기만 하면 뼈를 문겠습니다.

```
f = open("삼성전자 자소서.txt", "r", encoding="utf-8") # 삼성전자 자소서 불러오기
text = f.read()
f.close()

text = text.replace("삼성전자", "세동전자") # 텍스트에서 삼성전자를 세동전자로 바꾸기

f = open("세동전자 자소서.txt", "w", encoding="utf-8") # 세동전자 자소서 만들기
f.write(text) # 세동전자 자소서 내용 저장
f.close()
```

한국어를 쓰거나 읽으려면  
`encoding="utf-8"`을 사용해야 됩니다.