

따라하며 배우는

# 파이썬과 데이터 과학



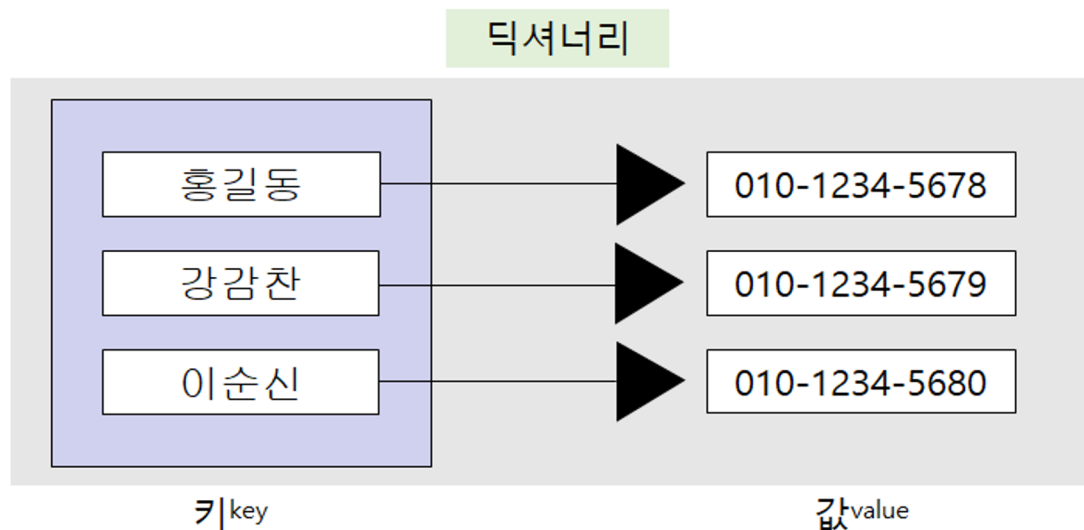
8장 연관된 데이터를  
딕셔너리로 짝을 짓자

## 이장에서 배울 것들

- 키를 이용해서 값을 추출할 수 있는 딕셔너리에 대하여 알아 보아요.
- 딕셔너리의 다양한 기능과 메소드에 대하여 알아 보아요.
- 딕셔너리를 사용하여 여러가지 문제를 해결해 보아요.
- 자료값의 중복을 허용하지 않는 순서가 없는 자료형인 집합에 대하여 알아 보아요.
- 합집합, 교집합 등 집합에 적용할 수 있는 편리한 연산을 알아 보아요.
- 집합을 이용하여 중복된 자료를 편리하게 제거하는 방법을 알아 보아요.
- 파이썬의 강력한 자료구조를 사용한 문제해결 기법을 익혀 보아요.

## 8.1 키와 값을 가진 딕셔너리로 자료를 저장하자

- 딕셔너리 `dictionary`도 리스트와 같이 값을 저장하는 자료구조로 파이썬에서는 기본 자료형으로 제공되고 있다.
- 하지만 딕셔너리에는 `값` `value`과 관련된 `키` `key`가 있다는 것이 큰 차이점이다.



## 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.

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

- 공백 리스트는 대괄호 []로 생성하고, 딕셔너리는 중괄호 {}로 생성한다는 것에 유의하자.

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

## 8.1 키와 값을 가진 딕셔너리로 자료를 저장하자

- 지금까지 작성한 것을 출력하여 보자.

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

- phone\_book 딕셔너리를 출력하면 딕셔너리의 항목item이 쉼표로 구분되어 출력된다.
- 딕셔너리를 생성하면서 동시에 초기화하는 방법도 있다.

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

- 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

- 리스트와 마찬가지로 딕셔너리에는 어떤 유형의 값도 저장할 수 있다.
- 아래 코드를 살펴보면 두 번째 항목에서 'Age'라는 키와 정수 27이 이 키의 값으로 사용되고 있다.

```
person_dic = {'Name': '홍길동', 'Age': 27, 'Class': '초급'}
```

```
print(person_dic['Name'])    # 딕셔너리의 'Name'이라는 키로 값을 조회함
```

```
print(person_dic['Age'])     # 딕셔너리의 'Age'이라는 키로 값을 조회함
```

person\_dic

홍길동

27

Name

‘홍길동’

Age

27

Class

‘초급’

## 8.2 딕셔너리의 기능을 알아보자

- 딕셔너리에서 가장 중요한 연산은 키를 가지고 연관된 값을 찾는 것
- 주소록이 있을 경우 사람 이름을 가지고 전화번호를 찾을 수 있어야 할 것이다.

```
>>> print(phone_book["강감찬"])  
010-1234-5679
```

- 리스트에서는 인덱스를 가지고 항목을 찾을 수 있지만 딕셔너리에서는 키가 있어야 값을 찾을 수 있다.
- 딕셔너리에서 사용되는 모든 키를 출력하려면 다음과 같이 keys()라는 메소드를 사용한다.

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

keys() 메소드를  
사용하면 키를 얻을  
수 있다.

## 8.2 딕셔너리의 기능을 알아보자

- 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.
- `for`문에서 `phone_book.item()`와 같은 함수를 호출하면 (키, 값) 튜플이 반환되므로 아래처럼 활용할 수 있다.

```
>>> 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.



## 8.2 딕셔너리의 기능을 알아보자

### 도전문제 8-1



#### 도전문제 8.1

여러분의 핸드폰에 있는 다음과 같은 연락처 정보를 `contact`라는 이름의 딕셔너리 구조로 표현해 보자. 여러분의 정보를 그림과 같은 키-값을 가지도록 항목을 생성하도록 하자. 그리고 나서 이 정보를 **for** 문을 사용해서 출력해 보도록 하자.

## 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 정렬할 수 있다.
- 람다 표현식은 `x`를 인자로 받아 `x`의 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')]
```

- 만일 딕셔너리의 항목을 삭제하려면 다음과 같이 `del`을 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

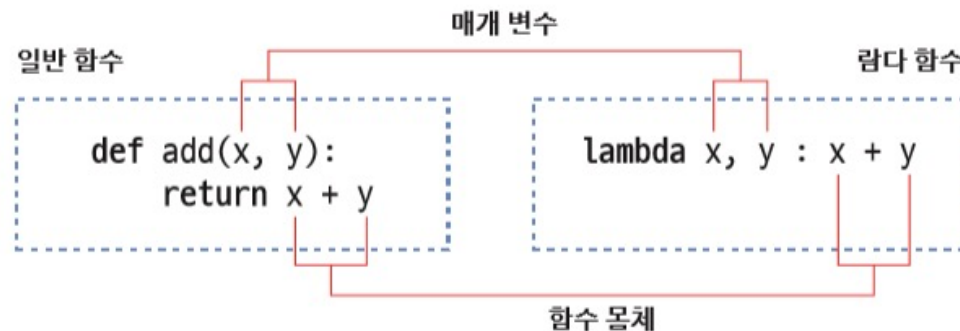
메소드	하는 일
<code>keys()</code>	딕셔너리 내의 모든 키를 반환한다.
<code>values()</code>	딕셔너리 내의 모든 값을 반환한다.
<code>items()</code>	딕셔너리 내의 모든 항목을 [키]:[값] 쌍으로 반환한다.
<code>get(key)</code>	키에 대한 값을 반환한다. 키가 없으면 None을 반환한다.
<code>pop(key)</code>	키에 대한 값을 반환하고, 그 항목을 삭제한다. 키가 없으면 <code>KeyError</code> 예외를 발생시킨다.
<code>popitem()</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 expression`이라고도 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 lamda 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')]
```



## LAB<sup>8-1</sup> 편의점 재고 관리 프로그램을 만들자

편의점에서 재고 관리를 수행하는 프로그램을 작성해보자. 이를 위해서 편의점에서 판매하는 물건의 재고를 딕셔너리에 저장한다. 그리고 사용자로부터 물건의 이름을 입력받아서 물건의 재고를 출력하는 프로그램을 작성해보자.  
아주 작은 편의점이라 취급하는 물건은 다음과 같다고 가정하자.



```
items = { "커피음료": 7, "펜": 3, "종이컵": 2, "우유": 1, "콜라": 4, "책": 5 }
```

### 원하는 결과

물건의 이름을 입력하시오: 콜라  
재고 : 4

## LAB<sup>8-1</sup> 편의점 재고 관리 프로그램을 만들자

```
items = { "커피음료": 7, "펜": 3, "종이컵": 2, "우유": 1, "콜라": 4, "책": 5 }  
  
name = input("물건의 이름을 입력하시오: ")  
print('재고 :', items[name])
```

## 8.4 람다 함수 = 이름이 없는 함수



### 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.

## LAB<sup>8-2</sup> 영한 사전을 만들어 보자

영한 사전과 같이 영어 단어를 주면, 이에 해당하는 우리말 단어를 알 수 있게 하려고 한다. 딕셔너리 구조를 활용하여 단어를 입력하고, 검색할 수 있는 프로그램을 작성해 보라. 실행된 결과는 명령 프롬프트 '\$'가 나타나며, 입력 명령은 '<', 검색 명령은 '>'로 표현 한다. 입력은 "영어 단어:우리말 단어" 형태로 이루어지며, 검색은 영어 단어를 입력한다. 프로그램의 종료는 'q' 키를 입력한다.

### 원하는 결과

사전 프로그램 시작... 종료는 q를 입력

\$ < one:하나

\$ < two:둘

\$ < house:집

\$ < Korea:한국

\$ > one

하나

\$ > house

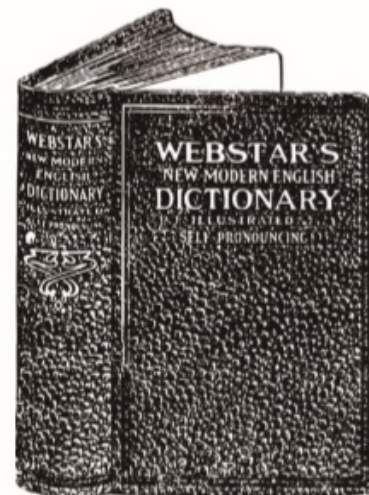
집

\$ > body

body가 사전에 없습니다.

\$ q

사전을 종료합니다.



## LAB<sup>8-2</sup> 영한 사전을 만들어 보자

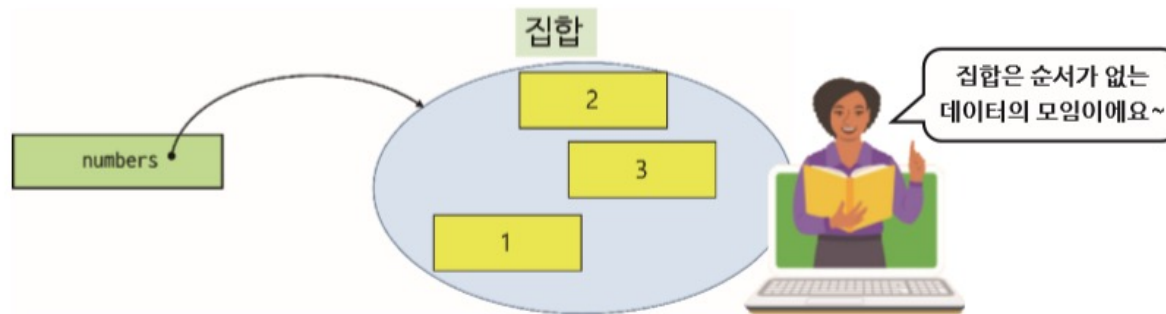
```
print("사전 프로그램 시작... 종료는 q를 입력")
dictionary = {}

while True:
    st = input('$ ')
    command = st[0]          # 첫 입력 문자를 추출한다
    if command == '<':
        st = st[1:]
        inputStr = st.split(':')
        if len(inputStr) < 2 :
            print('입력 오류가 발생했습니다.')
        else:
            dictionary[inputStr[0].strip()] = inputStr[1].strip()
    elif command == '>':
        st = st[1:]
        inputStr = st.strip()
        if inputStr in dictionary:
            print(dictionary[inputStr])
        else :
            print('{}가 사전에 없습니다.'.format(inputStr))
    elif command == 'q':
        break
    else :
        print('입력 오류가 발생했습니다.')
print("사전 프로그램을 종료합니다.")
```

## 8.5 명확한 기준을 가진 대상들이 모이면 : 집합

- 파이썬에서 사용하는 **집합** `set` 역시 수학의 집합과 비슷하며, 튜플과 달리 순서가 없는 자료형이다.
- 특히 동일한 값을 가지는 항목의 중복이 허용되지 않는다는 특징이 있으며, 교집합, 합집합, 차집합, 대칭차집합 등의 다양한 집합 연산을 수행할 수 있다.

```
>>> numbers = {2, 1, 3}    # 숫자 3개로 이루어진 집합 자료형
>>> numbers
{1, 2, 3}
```



## 8.5 명확한 기준을 가진 대상들이 모이면 : 집합

- 다음과 같이 리스트로부터 집합을 생성하는 것도 가능하다.
- 하지만 집합자료형에서는 요소가 중복되면 자동으로 중복된 요소를 제거한다.

```
>>> set([1, 2, 3, 1, 2])    # 리스트로부터 집합을 생성함  
{1, 2, 3}
```

- 그리고 문자열으로부터 집합을 생성하는 것도 가능한데, 이때는 각 문자들이 하나의 요소가 된다.

```
>>> set("abcdefa")         # 문자열도 시퀀스형이라서 집합형으로 변환이 가능하다  
{'f', 'a', 'b', 'e', 'c', 'd'}
```

- 비어 있는 집합을 생성하려면 다음과 같이 set() 함수를 사용한다.

```
>>> numbers = set()       # 비어있는 집합 생성
```

## 8.6 집합의 항목에 접근하는 연산

- 어떤 항목이 집합 안에 있는지를 검사하려면 리스트와 마찬가지로 `in` 연산자를 사용하면 된다.

```
numbers = {2, 1, 3}
if 1 in numbers:    # 1이라는 항목이 numbers 집합에 있는가 검사
    print("집합 안에 1이 있습니다.")
```

집합 안에 1이 있습니다.

- 집합의 항목은 순서가 없기 때문에 인덱스를 가지고 접근할 수는 없다.
- 하지만 `for` 반복문을 이용하여 각 항목들에 접근할 수 있다.

```
numbers = {2, 1, 3}
for x in numbers:
    print(x, end=" ")
```

1 2 3



## 8.6 집합의 항목에 접근하는 연산

- 여기서 주의할 점은 항목들이 출력되는 순서는 입력된 순서와 다를수도 있다는 점이다.
- 만약 정렬된 순서로 항목을 출력하기를 원한다면 다음과 같이 `sorted()` 함수를 사용하면 된다.

```
for x in sorted(numbers):  
    print(x, end=" ")
```

```
1 2 3
```

## 8.6 집합의 항목에 접근하는 연산

- 집합의 요소에는 인덱스가 없기 때문에 인덱싱이나 슬라이싱 연산은 의미가 없다.
- 우리는 `add()` 메소드를 이용하여서 하나의 요소를 추가할 수 있다.

```
>>> numbers = {1, 2, 3}
>>> numbers.add(4)      # numbers 집합에 원소 4를 추가
>>> numbers
{1, 2, 3, 4}
```

- 또한 집합의 요소를 삭제할 때는 `remove()` 메소드를 사용할 수 있다.

```
>>> numbers.remove(4)   # numbers 집합에 원소 4를 삭제
>>> numbers
{1, 2, 3}
```

## 8.6 집합의 항목에 접근하는 연산



### 잠깐 - in 연산자는 집합에만 사용하는 것은 아니다

이 절에서 사용한 **in** 연산자는 집합에만 사용할 수 있는 연산자가 아니라 여러 항목을 가진 다양한 데이터에 적용할 수 있다. 대표적인 예로 리스트에 특정한 항목이 있는지도 다음과 같이 검사할 수 있다.

```
>>> a_list = ['hello', 'world', 'welcome', 'to', 'python']
>>> 'python' in a_list    # 'python' 문자열 항목이 a_list에 있는가 조사한다
True
```

## 8.7 집합에 적용할 수 있는 다양한 연산들

### 집합 비교 연산

- 2개의 집합이 서로 같은지도 검사할 수 있다.
- 이것은 `==`과 `!=` 연산자를 사용하는 것이 가장 쉽다.

```
>>> A = {1, 2, 3}
>>> B = {1, 2, 3}
>>> A == B          # A가 B의 같은지 검사
True
```

- `<` 연산자와 `<=` 연산자를 사용하면 집합이 진부분집합인지, 부분집합인지를 검사할 수 있다.

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> B < A           # B가 A의 진부분 집합인가 검사
True
```

## 8.7 집합에 적용할 수 있는 다양한 연산들

### 항목들에 대한 정보 처리

- 집합에 대해서도 `len()`, `max()`, `min()`, `sorted()`, `sum()` 등의 메소드는 사용할 수 있다.
- 아래와 같이 6개의 항목을 가지고 집합을 만들어 보자.

```
>>> a_set = {1, 5, 4, 3, 7, 4 }    # 6개 항목으로 집합 생성
>>> len(a_set)                    # 항목의 개수는 중복을 제외하면 5개이다
5
>>> max(a_set)                    # 항목 가운데 가장 큰 수는 7
7
>>> min(a_set)                    # 항목 가운데 가장 작은 수는 1
1
>>> sorted(a_set)                 # 항목을 정렬하여 리스트 만든다. 집합이므로 중복은 제거한다
[1, 3, 4, 5, 7]
>>> sum(a_set)                    # 중복 원소는 하나만 사용되므로 전체 합은 20
20
```

## 8.7 집합에 적용할 수 있는 다양한 연산들

### 집합에 적용할 수 있는 논리 연산

- 다수의 항목을 가진 데이터에 대해 적용할 수 있는 논리 연산으로 `all()`과 `any()` 함수가 있다.
- 이 함수는 인자로 주어진 데이터의 항목들 각각에 대해 부울형 평가를 한 결과를 종합적으로 반환한다.

```
>>> a_set = { 1, 0, 2, 3, 3}
>>> all(a_set)    # a_set이 모두 True인가를 검사한다
False
>>> any(a_set)    # a_set에 0이 있는가를 검사한다
True
```

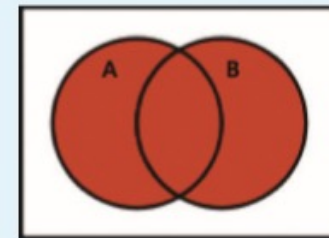
## 8.8 풍부하고 멋진 집합 연산

- 집합이 유용한 이유는 교집합이나 합집합과 같은 여러가지 집합 연산을 지원하기 때문이다.
- 이것은 연산자나 메소드로 수행할 수 있다.

```
>>> A = {1, 2, 3}
>>> B = {3, 4, 5}
```

- 합집합은 다음과 같이 2개의 집합을 합하는 연산으로 | 연산자나 union() 메소드를 사용한다.

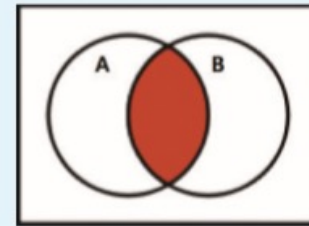
```
>>> A | B      # 합집합 연산
{1, 2, 3, 4, 5}
>>> A.union(B) # 합집합 메소드
{1, 2, 3, 4, 5}
```



## 8.8 풍부하고 멋진 집합 연산

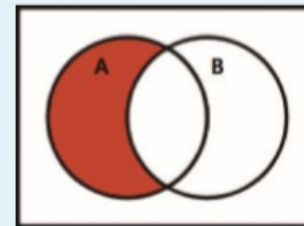
- 교집합은 2개의 집합에서 겹치는 요소를 구하는 연산이다.
- 교집합은 `&` 연산자나 `intersection()` 메소드를 사용한다.

```
>>> A & B      # 교집합 연산
{3}
>>> A.intersection(B) # 교집합 메소드
{3}
```



- 차집합은 하나의 집합에서 다른 집합의 요소를 빼는 것이다.
- 차집합은 `-` 연산자나 `difference()` 메소드를 사용한다.

```
>>> A - B      # 차집합 연산
{1, 2}
>>> A.difference(B) # 차집합 메소드
{1, 2}
```

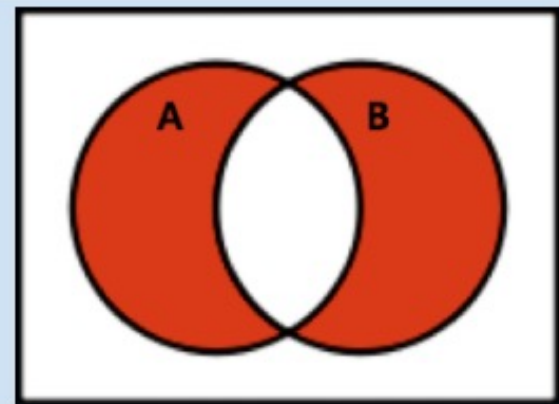




## 8.8 풍부하고 멋진 집합 연산

- 대칭차집합은 두 집합의 합집합에서 교집합을 뺀 요소를 구하는 연산이다.
- 대칭차집합은  $\wedge$  연산자나 `symmetric_difference()` 메소드를 사용한다.

```
>>> A ^ B                # 대칭 차집합 연산
{1, 2, 4, 5}
>>> A.symmetric_difference(B)
{1, 2, 4, 5}
```



## LAB<sup>8-3</sup> 파티 동시 참석자 알아내기

파티에 참석한 사람들의 명단이 집합 partyA와 partyB에 각각 저장되어 있다.

```
partyA = set(["Park", "Kim", "Lee"])  
partyB = set(["Park", "Choi"])
```

2개 파티에 모두 참석한 사람들의 명단을 출력하려면 어떻게 해야 할까?

### 원하는 결과

2개의 파티에 모두 참석한 사람은 다음과 같습니다.  
{'Park'}



## LAB<sup>8-3</sup> 파티 동시 참석자 알아내기

```
partyA = set(["Park", "Kim", "Lee"])
partyB = set(["Park", "Choi"])

print("2개의 파티에 모두 참석한 사람은 다음과 같습니다. ")
print ( partyA.intersection(partyB))
```

## 8.8 풍부하고 멋진 집합 연산

### 도전문제 8-3



#### 도전문제 8.3

LAB 8-3의 데이터를 이용하여 다음과 같이 파티 A, 파티 B에 참석한 사람들을 중복되지 않도록 다음과 같이 출력하자.

파티 A, B에 참석한 사람들 : Park, Kim, Lee, Choi

LAB 8-3의 데이터를 이용하여 다음과 같이 파티 A, 파티 B중 한군데만 참석한 사람들을 출력하자.

파티 A, B에 중복없이 참석한 사람 : Kim, Lee, Choi

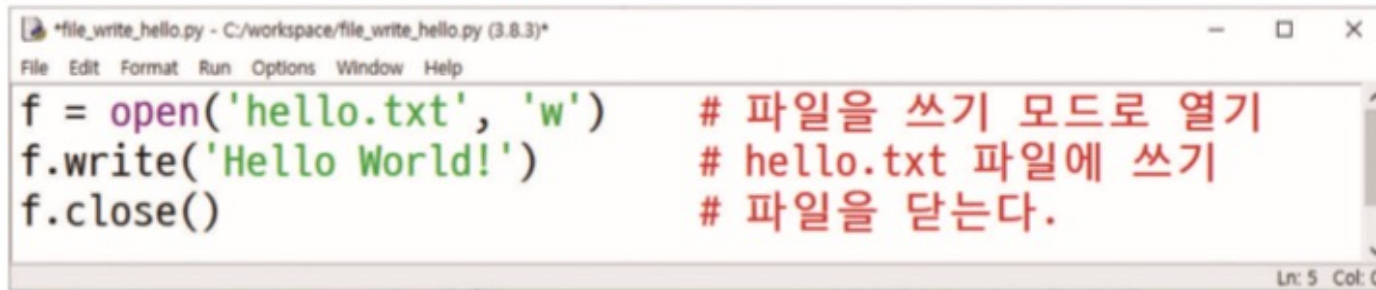
LAB 8-3의 데이터를 이용하여 다음과 같이 파티 A에만 참석한 사람들을 출력하자.

파티 A에만 참석한 사람 : Kim, Lee

## 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.
- 우선 다음과 같은 내용의 코드를 작성하고 C: \ workspace 아래에 file\_write\_hello.py 라는 이름으로 저장하도록 하자.

## 8.9 파일로부터 자료를 읽고 저장해보자



```
*file_write_hello.py - C:/workspace/file_write_hello.py (3.8.3)*
File Edit Format Run Options Window Help
f = open('hello.txt', 'w')    # 파일을 쓰기 모드로 열기
f.write('Hello World!')      # hello.txt 파일에 쓰기
f.close()                    # 파일을 닫는다.
Ln: 5 Col: 0
```

- 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.

## 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 for문을 사용할 수 있다.

## LAB<sup>8-4</sup> 파일에서 중복되지 않은 단어의 개수 구하기

작문을 할 때 다양한 단어를 사용하면 높은 점수를 받는다. 텍스트 파일을 읽어서 단어를 얼마나 다양하게 사용하여 문서를 작성하였는지를 계산하는 프로그램을 작성해보자. 중복된 단어는 하나만 인정한다. 집합은 중복을 허용하지 않기 때문에 단어를 집합에 추가하면 중복되지 않은 단어가 몇 개나 사용되었는지를 알 수 있다.

proverb.txt에 다음과 같은 내용이 담겨 있다고 가정하자(github에 있음).

```
All's well that ends well.  
Bad news travels fast.  
Well begun is half done.  
Birds of a feather flock together.
```

### 원하는 결과

입력 파일 이름: proverb.txt

사용된 단어의 개수 = 18

```
{'travels', 'half', 'that', 'news', 'alls', 'well', 'fast', 'feather', 'flock',  
'bad', 'together', 'ends', 'is', 'a', 'done', 'begun', 'birds', 'of'}
```



## LAB<sup>8</sup>-4 파일에서 중복되지 않은 단어의 개수 구하기

```
# 단어에서 구두점을 제거하고 소문자로 만든다.
def process(w):
    output = ""
    for ch in w:
        if ch.isalpha() :
            output += ch
    return output.lower()

words = set() # 중복을 방지하기 위해 집합 자료형에 단어를 넣자
fname = input("입력 파일 이름: ")
file = open(fname, "r") # 파일을 연다.
# 파일의 모든 줄에 대하여 반복한다.
for line in file:
    lineWords = line.split()
    for word in lineWords:
        words.add(process(word)) # 단어를 집합에 추가한다.

print("사용된 단어의 개수 =", len(words))
print(words)
```

## 8.10 두 수의 약수와 최대공약수 그리고 프로그래밍적인 사고

- 두 정수의 약수를 구하는 방법을 생각해 보자.
- 10의 약수는 1, 2, 5, 10 이 될 수 있는데 이 중에서 1과 자기 자신인 10을 제외한 2와 5를 진약수라고 한다.
- 다음과 같이 리스트를 만들고 2, 3, 4, 5..9까지의 모든 수를 10으로 나누어 그 나머지 값이 0이 되는지를 확인해보고 그렇다면 리스트에 담아 두면 된다.

```
num = 10
divisors = []

for i in range(2, num):
    if num % i == 0:
        divisors.append(i)

print(num, '의 진약수 :', divisors)
```

```
10 의 진약수 : [2, 5]
```

## 8.10 두 수의 약수와 최대공약수 그리고 프로그래밍적인 사고

- 이 코드를 활용하여 48과 60의 최대공약수를 구하려면 48과 60의 진약수를 구한 다음 공통된 값을 구하고 그 중에서 가장 큰 값을 찾는 방식으로 최대공약수를 구해보자.
- 이를 위하여 `get_divisors()` 함수 내에 `divisors`라는 빈 집합 자료형을 선언하고 `num % i`가 0이 되는 값을 이 집합에 추가하도록 하자.

```
def get_divisors(num):    # num의 약수를 집합형으로 반환함
    divisors = set()
    for i in range(2, num):
        if num % i == 0:
            divisors.add(i)
    return divisors

x = 48
print(x, '의 진약수 :', get_divisors(x))
y = 60
print(y, '의 진약수 :', get_divisors(y))
```

```
48 의 진약수 : {2, 3, 4, 6, 8, 12, 16, 24}
60 의 진약수 : {2, 3, 4, 5, 6, 10, 12, 15, 20, 30}
```

## 8.10 두 수의 약수와 최대공약수 그리고 프로그래밍적인 사고

- 이제 이 약수들 중에서 공통된 약수를 구해보자.
- 그리고 이 값들 중 가장 큰 값을 `max()` 함수를 이용해서 구해보자.

```
A = get_divisors(x)
B = get_divisors(y)

print(A.intersection(B))
print(x, y, '의 최대공약수 :', max(A.intersection(B)))
```

```
{2, 3, 4, 6, 12}
48 60 의 최대공약수 : 12
```

- 물론 이 방법보다 더 빠르게 두 수의 최대공약수를 구하는
- 유클리드의 방법이 있으나, 이와 같이 문제를 해결하는 단계적 과정을 통해서 해를 구하는 것이 바로 프로그래밍적인 사고 방식일 것이다.



summary

## 핵심 정리



- 딕셔너리는 키와 값으로 이루어진다.
- 딕셔너리에 키를 제시하면 값을 반환한다.
- 딕셔너리는 인덱스를 이용하여 접근하지 않고 키를 이용하여 항목에 접근한다.
- 키를 이용하기 때문에 항목들 사이의 순서는 중요하지 않다.
- 항목의 순서가 중요하지 않은 데이터로 집합이 있다.
- 딕셔너리와 집합은 항목의 순서에 의미가 없으므로 슬라이싱을 적용할 수 없다.
- 집합연산을 지원하는 풍부한 연산자와 메소드가 제공되고 있다.

따라하며 배우는

# 파이썬과 데이터 과학



Questions?