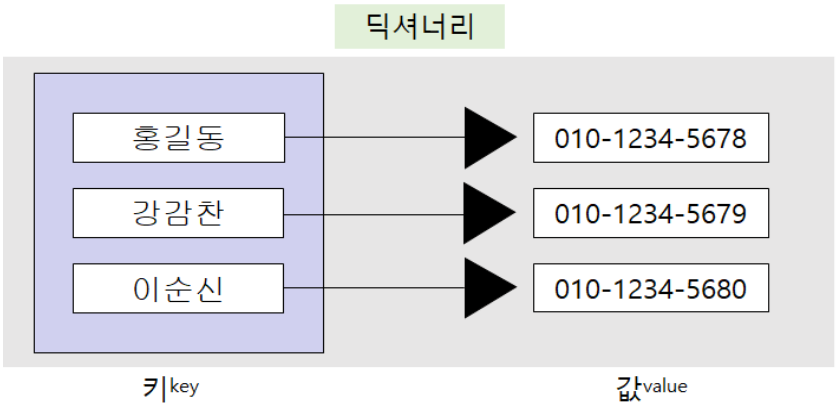
# 데이터분석 프로그래밍
## 딕셔너리

임현기

# 딕셔너리

- 딕셔너리
  - 파이썬에서 제공하는 기본 자료구조
  - 키와 값으로 데이터 저장 (키-값 쌍)




# 딕셔너리

- 딕셔너리 생성: {}

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

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

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

- 딕셔너리 생성과 초기화

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

```
>>> 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'}
```

# 딕셔너리 기능

```
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
```

```
홍길동
27
```

person_dic

| Name | '홍길동' |
|---|---|
| Age | 27 |
| Class | '초급' |

# 딕셔너리 기능

- 키를 가지고 값을 찾는 것: 딕셔너리의 가장 중요한 기능

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

- keys() 메소드를 통해 모든 키 출력 가능

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

keys() 메소드를 사용하면 키를 얻을 수 있다.

# 딕셔너리 기능

- values() 메소드로 모든 값 출력 가능

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

- items() 메소드로 모든 키-값을 출력 가능

- for문으로 튜플 반환

```
>>> 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.

# 딕셔너리 기능

- for문 + keys()

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

- sorted 메소드를 통한 딕셔너리 정렬

```
>>> 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')]
```

- del 메소드를 이용한 삭제

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

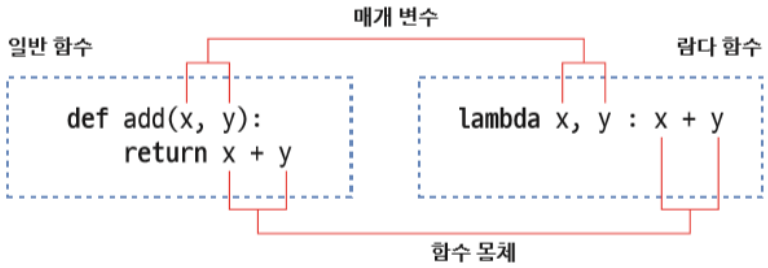
# 딕셔너리 기능

| 메소드 | 하는 일 |
|---|---|
| keys() | 딕셔너리 내의 모든 키를 반환한다. |
| values() | 딕셔너리 내의 모든 값을 반환한다. |
| items() | 딕셔너리 내의 모든 항목을 [키]:[값] 쌍으로 반환한다. |
| get(key) | 키에 대한 값을 반환한다. 키가 없으면 None을 반환한다. |
| pop(key) | 키에 대한 값을 반환하고, 그 항목을 삭제한다. 키가 없으면 KeyError 예외를 발생시킨다. |
| popitem() | 제일 마지막에 입력된항목을 반환하고 그 항목을 삭제한다. |
| clear() | 딕셔너리 내의 모든 항목을 삭제한다. |

# 람다 함수

- 람다 표현식(람다 함수)
  - 함수를 만들지 않고 함수화된 기능만 불러 사용
  - 표현식 안에 새로운 변수 선언할 수 없음
  - 한 줄로만 표현 가능

# 람다 함수




- 두 값의 합을 구하는 람다 함수

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

- 첫 항목만 추출하는 람다 함수

```
>>> 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
```

(100, 200, 300)

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

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

# 람다 함수

- x[0]로 반환하면 키로 정렬
- x[1]로 반환하면 값으로 정렬

```
>>> 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')]
```

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
```

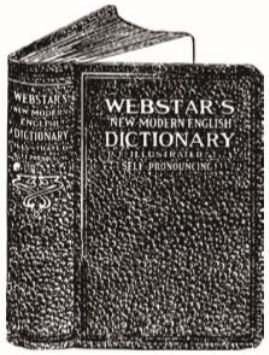
```
items = { "커피음료": 7, "펜": 3, "종이컵": 2, "우유": 1, "콜라": 4, "책": 5 }

name = input("물건의 이름을 입력하시오: ")
print('재고 :', items[name])
```

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.
```



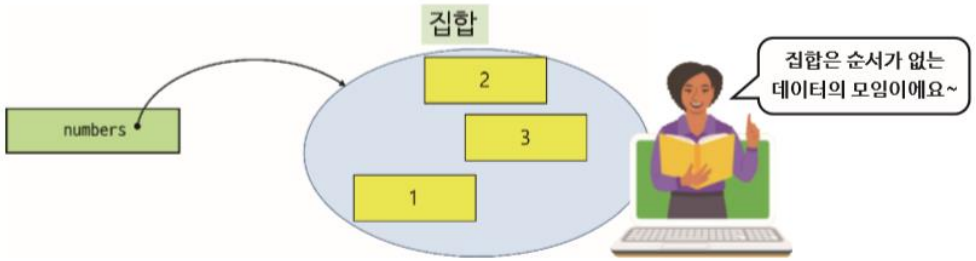
```
print("사전 프로그램 시작... 종료는 q를 입력")
dictionary = {}

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.")
```

# 집합

- 순서가 없는 자료형
  - 중복 허용하지 않음
  - 교집합, 합집합, 차집합, 대칭차집합 등 집합 연산 수행 가능

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



# 집합

- 리스트로부터 집합 생성 가능
  - 요소가 중복되면 자동으로 제거

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

- 문자열로 집합 생성

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

- 비어있는 집합 생성

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

# 집합 연산

- 항목 검색: in 연산자 사용

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

```
집합 안에 1이 있습니다.
```

- 인덱스로 접근 불가
  - for문을 이용한 접근은 가능
  - 입력된 순서와 다를 수 있음

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

```
1 2 3
```

  - sorted() 메소드를 사용하면

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

```
1 2 3
```

# 집합 연산

- 인덱스가 없기 때문에 슬라이싱 연산은 무의미

- add() 메소드를 이용하여 요소 추가

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

- remove() 메소드로 요소 삭제

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

# 집합 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
```

# 집합 연산

- 비교 연산
  - 같은지 다른지 ==, !=

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

  - 진부분집합, 부분집합 <, <=

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

# 집합 연산

- 정보 처리
  - len()
  - max(), min()
  - sorted()
  - sum()

```
>>> 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
```

# 집합 연산

- 논리 연산
  - all(): 모두 참일 때 True
  - any(): 하나라도 참일 때 True

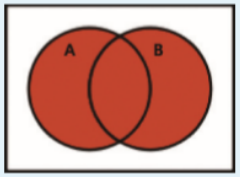
```
>>> a_set = { 1, 0, 2, 3, 3}
>>> all(a_set)   # a_set이 모두 True인가를 검사한다
False
>>> any(a_set)   # a_set에 0이 있는가를 검사한다
True
```

# 집합 연산

- 합집합
  - | 연산자
  - union() 메소드

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

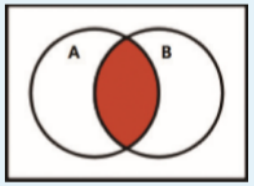
```
>>> A | B     # 합집합 연산
{1, 2, 3, 4, 5}
>>> A.union(B)    # 합집합 메소드
{1, 2, 3, 4, 5}
```



# 집합 연산

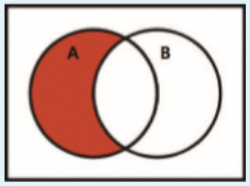
- 교집합
  - & 연산자
  - intersection() 메소드

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




- 차집합
  - - 연산자
  - difference() 메소드

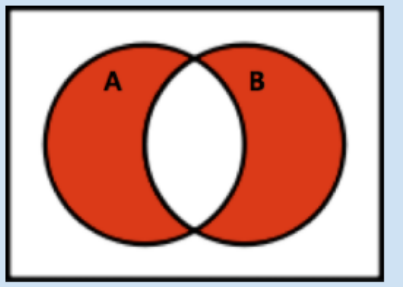
```
>>> A - B     # 차집합 연산
{1, 2}
>>> A.difference(B) # 차집합 메소드
{1, 2}
```



# 집합 연산

- 대칭차집합
  - ^ 연산자
  - symmetric_difference() 메소드

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



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'}
```

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

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

## 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
```

# 자료 읽고 쓰기

- 파일 쓰기


- 파일 읽기

```
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!
```

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'}
```