파이썬 기초: 반복문과 복합 자료형

#반복문 #복합자료형 #Tuple #Dictionary #Unpacking

목차

반복문

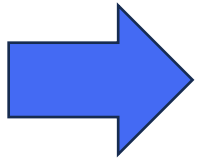
반복문의 필요성

쇼핑몰에서 장바구니에 담긴 상품의 가격 총 합계를 구하려는 상황..

```
prices = [3000, 4500, 10000, 8000, 22000]
quantities = [3, 4, 1, 2, 1]

total = 0

total += prices[0] * quantities[0]
total += prices[1] * quantities[1]
total += prices[2] * quantities[2]
total += prices[3] * quantities[3]
total += prices[4] * quantities[4]
```

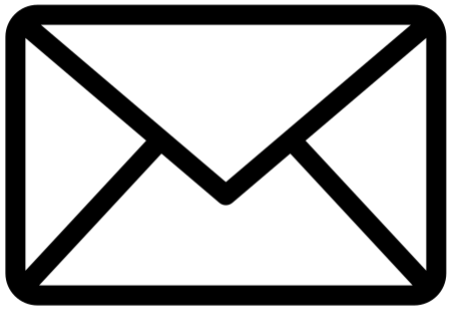


상품이 5개가 아니라 1000개..
그 이상이라면?...

반복문이란?

비슷한 명령을 반복하는 코드를 묶어서 표현하자!

어떠한 조건이나 범위 내에서 어떠한 명령을 반복적으로 수행하는 것



우리 회사의 서비스를 구독 중인 고객들에게
뉴스레터 보내기

회의 보고서 4부 출력하기

반복문: for문

시퀀스 자료형 방식의 for문

시퀀스에서

in 시퀀스

원소를 하나씩 가져와서

for 변수

명령 실행

명령

Python의 반복문 1: for문

시퀀스 자료형의 for문: 시퀀스 자료형의 원소를 하나씩 순회(iterate)

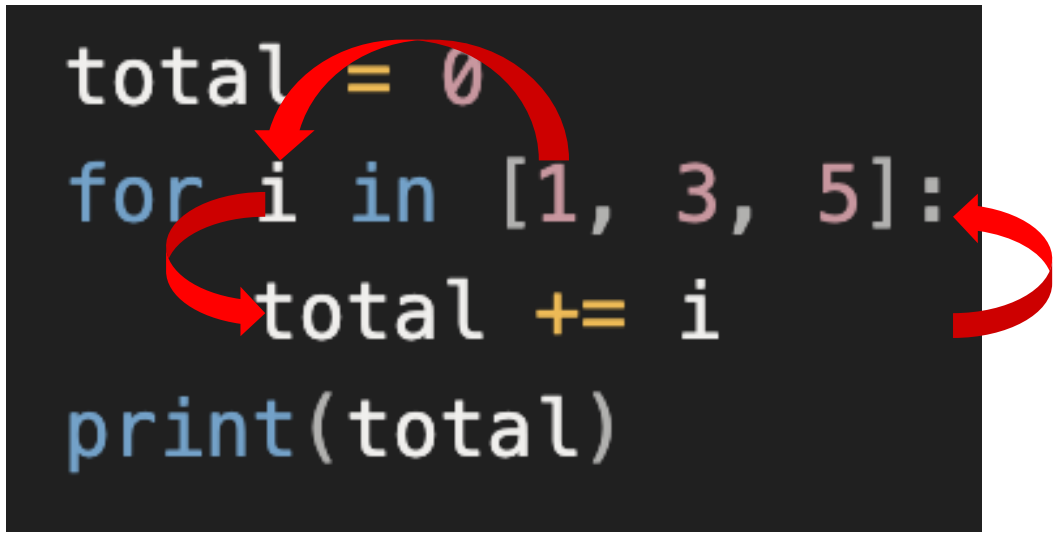
for문의 구조

```
for 변수 in 시퀀스:
    <수행할 명령>
    <수행할 명령>
    ...
```

if문과 동일하게 들여쓰기!

1. 시퀀스에서 앞에서부터 원소를 하나씩 꺼냄
2. 그 원소를 변수에 저장
3. 수행할 명령들 실행
4. 그 다음 원소를 꺼내고 시퀀스의 원소가 다 꺼내질 때까지 1~3번 반복

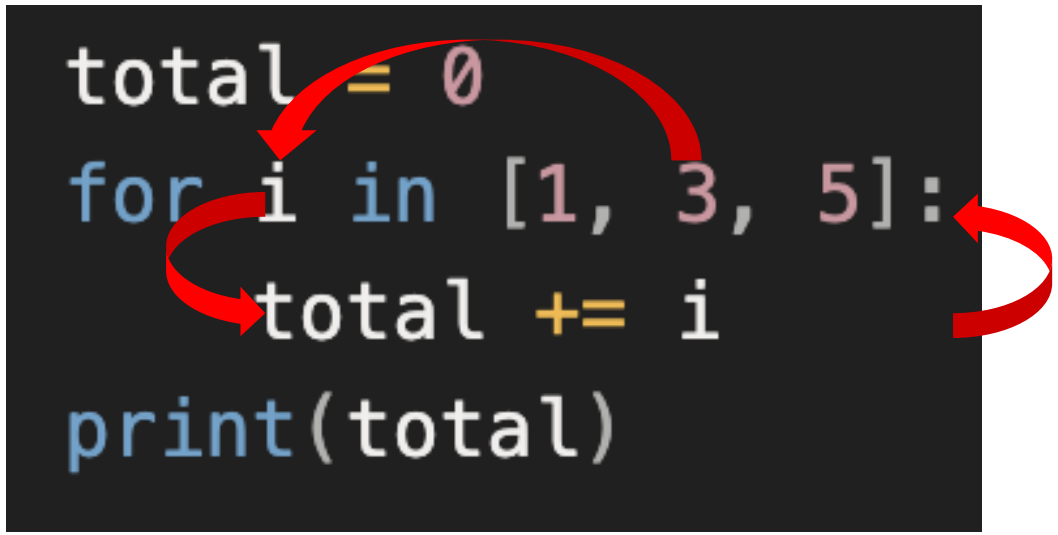
for문: Walkthrough Example: 첫번째 사이클



1. 첫번째 원소를 꺼내고, 변수 i에 저장
2. 명령 실행
3. 위로 이동

변수	값
total	1
i	1

for문: Walkthrough Example: 두번째 사이클



1. 두번째 원소를 꺼내고, 변수 i에 저장
2. 명령 실행
3. 위로 이동

변수	값
total	4
i	3

for문: Walkthrough Example: 세번째 사이클

```
total = 0
for i in [1, 3, 5]:
    total += i
print(total)
```

출력 결과

```
9
```

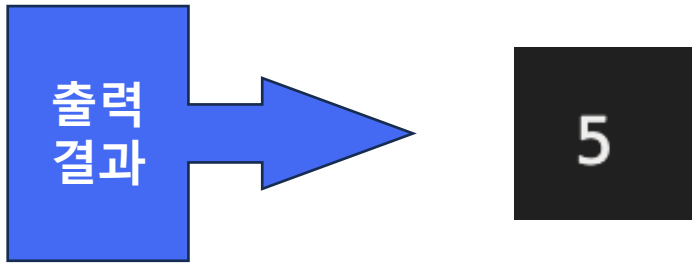
1. 세번째 원소를 꺼내고, 변수 i에 저장
2. 명령 실행
3. 위로 이동
4. 시퀀스의 모든 원소를 꺼냈으므로 for문 종료

변수	값
total	9
i	5

Python의 반복문 1: for문

for문의 반복 횟수? = 시퀀스 자료형의 길이(length)만큼

```
count = 0
for i in [1, 2, 3, 4, 5]:
    count += 1
print(count)
```



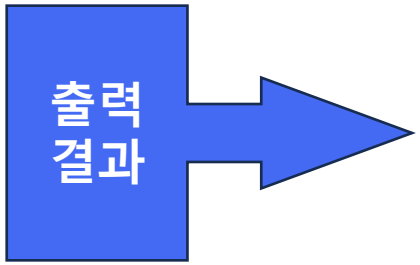
Python의 반복문 1: for문

Example

가격 목록에 각각 할인율 적용한 가격 출력 (10% 할인)

```
prices = [10000, 25000, 8000, 15000]
discount_ratio = 0.9

for price in prices:
    print(price * discount_ratio)
```




```
9000.0
22500.0
7200.0
13500.0
```

반복문: for-range문

for-range문

N회 동안

횟수 for

명령을 수행해라

명령

Python의 반복문 1: for-range문

range() 함수

연속되는 숫자를 만들어주는 함수
시작 숫자(a)와 끝 숫자(b)를 지정하면 a ~ (b-1) 까지의 숫자 생성

range(start, end, step)

```
print(range(1, 9))
print(list(range(1, 9)))

print(range(5))
print(list(range(5)))
```

출력 결과

```
range(1, 9)
[1, 2, 3, 4, 5, 6, 7, 8]
range(0, 5)
[0, 1, 2, 3, 4]
```

하나의 값만 넣으면 0 ~ (값-1)까지의 범위

Python의 반복문 1: for-range문

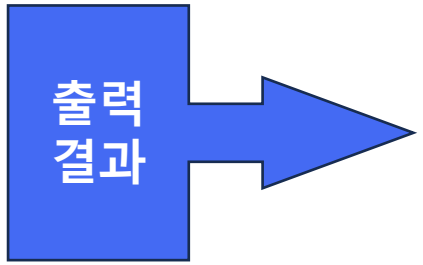
range() 함수

연속되는 숫자를 만들어주는 함수
시작 숫자(a)와 끝 숫자(b)를 지정하면 a ~ (b-1) 까지의 숫자 생성

range(start, end, step)

```
print(range(1, 9, 2))
print(list(range(1, 9, 2)))

print(range(1, 9, 3))
print(list(range(1, 9, 3)))
```




```
range(1, 9, 2)
[1, 3, 5, 7]
range(1, 9, 3)
[1, 4, 7]
```

step은 증감 간격

for-range문 (1): 구간으로 반복하기

a이상 b미만의 범위만큼 반복

```
for 변수 in range(a, b):
    <수행할 명령>
```

```
num_list = [1]

for i in range(2, 5):
    num_list.append(i)
print(num_list)
```

2부터 5 미만의 범위 반복

```
[1, 2, 3, 4]
```

for-range문 (2): 횟수로 반복하기

a번 만큼 반복

```
for 변수 in range(a):
    <수행할 명령>
```

```
count = 0

for i in range(10):
    count += 1
print(count)
```

10회 반복(= 0부터 10 미만의 범위)

```
10
```

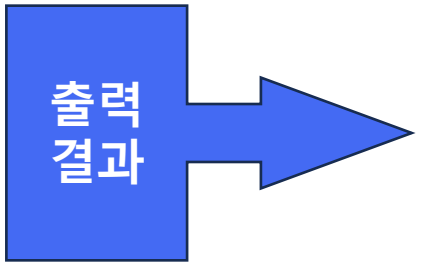
Python의 반복문 1: for-range문

for-range문 활용하기

len() 함수와 함께 활용하여, 인덱싱을 사용.

```
num_list = [3, 4, 1, 5, 2]

for i in range(len(num_list)):
    print(i, num_list[i])
```




```
0 3
1 4
2 1
3 5
4 2
```

두 가지 for문 비교하기

구분	시퀀스 방식	range() + len() 방식
접근	원소 값 직접 가져옴	인덱싱을 통해서 가져옴
코드	직관적	다소 복잡
사용 상황	값만 필요할 때	인덱스도 필요할 때
예시	for **fruit** in fruits: (실제 값)	for **i** in range(len(fruits)): (인덱스)

Python의 반복문 1: for-range문

Example

처음에 살펴봤던 예시를 구현해보면?!

```
prices = [3000, 4500, 10000, 8000, 22000]
quantities = [3, 4, 1, 2, 1]

total = 0
for i in range(len(prices)):
    total += prices[i] * quantities[i]
```

상품 개수가 많아져도 자동 확장

for문과 if문의 활용

상품의 가격이 10,000원 이상인 상품에 대해서만 10% 할인 적용

```
prices = [4000, 25000, 7500, 12000, 18500]
discounted = []

for i in range(len(prices)):
    if prices[i] >= 10000:
        discounted.append(prices[i] * 0.9)
    else:
        discounted.append(prices[i])

print(discounted)
```

출력 결과

```
[4000, 22500.0, 7500, 10800.0, 16650.0]
```

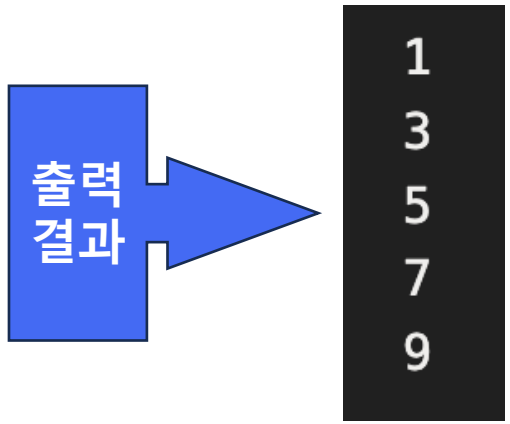
continue 키워드

반복문 내에서 continue를 만나면 나머지 명령을 건너뛰고, 다음 반복으로 넘어감

```
for i in range(1, 11):
    if i % 2 == 0:
        continue
    print(i)
```

짝수인 경우, continue를 만나기에
print(i)는 호출되지 않고, 다음 반복으로 이동

(특정 조건에 만나면 다음으로 스킵!)



반복문: while문

while문

조건이 참인

조건

동안

while

명령을 수행해라

명령

Python의 반복문 2: while문

while문

for문은 횟수 기반의 반복이라면, while문 조건 기반의 반복문

```
while 조건:
    <수행할 명령>
```

조건이 참일 동안 <수행할 명령>을 수행해라
조건이 아닐 때까지 반복해라

1. 조건을 검사 후 참(True)이면 명령 수행
2. 다시 위로 올라와서 조건 검사
3. 1 ~ 2번을 조건이 거짓(False)가 될 때까지 반복

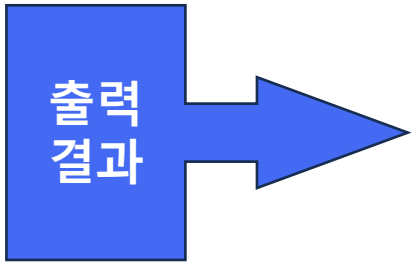
Python의 반복문 2: while문

while문

for문은 횟수 기반의 반복이라면, while문 조건 기반의 반복문

```
i = 5

while i > 0:
    print(i)
    i -= 1
print("while문 종료")
```




```
5
4
3
2
1
while문 종료
```

*i가 양수인 동안 반복
*i가 0보다 크지 않을 때까지 반복

Python의 반복문 2: while문

Example

계좌 잔액이 인출할 금액보다 클 때까지 인출하기

```
balance = 10000  # 초기 잔액
withdraw = 1500  # 인출 금액

while balance >= withdraw:
    print("현재 잔액:", balance)
    print(withdraw, "원 인출")
    balance -= withdraw

print("인출 종료. 최종 잔액:", balance)
```

출력 결과

```
현재 잔액: 10000
1500 원 인출
현재 잔액: 8500
1500 원 인출
현재 잔액: 7000
1500 원 인출
현재 잔액: 5500
1500 원 인출
현재 잔액: 4000
1500 원 인출
현재 잔액: 2500
1500 원 인출
인출 종료. 최종 잔액: 1000
```

Python의 반복문 2: while문

while문

조건의 반복에 사용되는 변수의 값을 바꾸는 코드가 있어야 한다

```
i = 5

while i > 0:
    print(i)
    i -= 1
print("while문 종료")
```

```
i = 0

while i < 5:
    print(i)
    i += 1
print("while문 종료")
```

그렇지 않으면?..

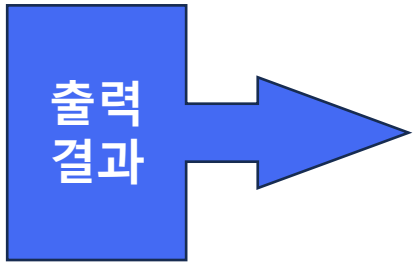
Python의 반복문 2: while문

무한 루프(infinite loop)

끝나지 않는 반복문: 조건이 항상 참(True)이 된다면?

```
i = 1

while i > 0:
    print(i)
    i += 1
print('종료')
```




```
...
7382382
7382383
7382384
7382385
7382386
...
```

무한루프가 발생하지 않게끔 코드를 작성해야 한다!

Python의 반복문 2: while문

무한 루프(infinite loop)을 탈출하려면?

조건문과 break를 활용!

break는 반복문을 **강제 종료**하는 역할 (break의 다음 코드는 실행되지 않고 즉시 탈출)

```
i = 1

while i > 0:
    print(i)

    if i >= 5:
        break
    i += 1
print('종료')
```

출력 결과

```
1
2
3
4
5
종료
```

i가 5가 될 때, 조건문이
참이 되어 break에 의해 탈출!

Python의 반복문 2: while문

Example

원하는 값 리스트에서 탐색하기

```
numbers = [3, 8, 12, 5, 18, 7]
target = 12
i = 0

while i < len(numbers):
    if numbers[i] == target:
        print("탐색 완료", target, i)
        break
    i += 1
```

출력 결과

```
탐색 완료 12 2
```

현재 원소가 target과 일치하는 순간에
break를 통해 탐색 종료

Python의 반복문 2: while문

일부러 무한 루프(infinite loop) 만들기

일부러 조건을 항상 참으로 만들고, 중간에 조건문+break 를 통해 탈출

```
while True:
    ...
    if 조건:
        break
    ...
```

사용자 입력 계속 받기 (메뉴 시스템)

```
while True:
    print("\n====== 쇼핑몰 메뉴 =====")
    print("1. 상품 담기")
    print("2. 장바구니 보기")
    print("3. 결제하기")
    print("4. 종료하기")
    choice = input("메뉴 선택: ")

    if choice == "1":
        # 상품 담기

    elif choice == "2":
        # 장바구니 보기

    elif choice == "3":
        # 결제하기

    elif choice == "4":
        print("프로그램 종료")
        break
    else:
        print("잘못된 입력입니다. 1~4 중에서 선택하세요.")
```

Tuple 자료형

Tuple 자료형의 필요성

대부분 여러 개의 값을 보관하기 위해서는 List 자료형을 사용

하지만, 리스트 자료형의 경우 **값이 바뀔 위험**이 있는 자료형이다.

```
num_list = [3, 1, 4, 5, 2]

num_list.append(6) # 값 추가
num_list[2] = 10 # 값 수정
num_list.remove(3) # 값 삭제
```

Tuple 자료형의 필요성

Why?

리스트 자료형으로 데이터를 관리해보는 상황

서울의 좌표와 부산의 좌표

```
seoul_location = [37.5665, 126.9780]
busan_location = [35.1796, 129.0756]
```

색상(RGB) 값

```
red = [255, 0, 0]
```

누군가의 실수로 인해 값이 바뀐다면?..

```
seoul_location.append(10)
```

```
red[0] = 20000
```

?

Tuple 자료형

값을 바꿀 수 없으면서, 여러 개의 값을 담을 수 있는 순서가 있는 자료형(read-only 구조)

소괄호와 콤마(,)를 활용하여 표현.

튜플 정의하기

```
tuple_zero = ()
tuple_one = (1,)
tuple_multiple = (1, 2, 3, 4, 5)
tuple_no_parentheses = 1, 2, 3, 4, 5
```

실질적으로 투플을 구분하는 것은 콤마(,)이다.

하지만 때에 따라 가독성을 위해 소괄호를 사용

Tuple 자료형

Tuple도 시퀀스 자료형!

Tuple도 시퀀스 자료형이기에, 특징을 모두 가지고 있다.

인덱싱과 슬라이싱

```
tuple_ex = (3, 2, 4, 5, 1)

print(tuple_ex[2])
print(tuple_ex[1:4])
```

in 연산자와 len() 함수

```
tuple_ex = (3, 2, 4, 5, 1)

print(5 in tuple_ex)
print(len(tuple_ex))
```

이어붙이기(+), 반복(*)도 가능

Tuple 자료형

Tuple은 읽기 전용(read-only) 구조

데이터를 추가/삭제/변경이 불가능하다. 한 번 생성되면 고정되는 자료형

```
tuple_ex = (3, 2, 4, 5, 1)

tuple_ex.append(6)

tuple_ex.remove(3)

tuple_ex[3] = 10
```

데이터 추가 ✖

데이터 삭제 ✖

데이터 변경 ✖

```
AttributeError: 'tuple' object has no attribute 'append'

AttributeError: 'tuple' object has no attribute 'remove'

TypeError: 'tuple' object does not support item assignment
```

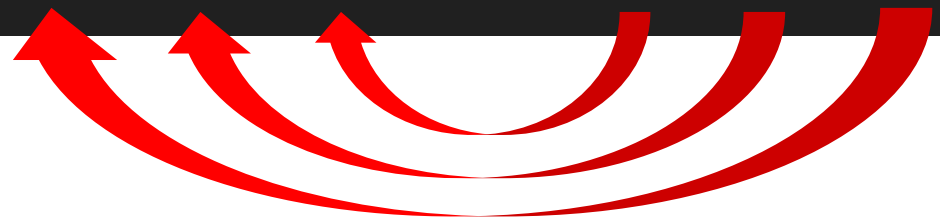
오류 발생!

복합 자료형 활용

Unpacking(언패킹)

묶여 있는 자료형(List, Tuple, Dictionary 등)의 원소들을 분해해서 각 변수들에 나누는 것

```
a, b, c = (1, 2, 3)
```



스왑, 함수 가변 매개변수, 함수 리턴, 반복문 등
파이썬에서 많이 활용된다!

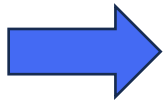
반대? Packing(패킹): 여러 값을 묶어서 하나의 변수(투플, 리스트 등)에 저장

```
packed = (1, 2, 3)
```

Unpacking 활용 (1): Swap(스왑)

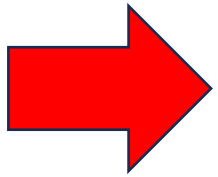
Swap: 두 변수 값을 서로 바꾸는 것

```
a = 10
b = 15
```



```
a = b
b = a
```

?



언패킹을 활용하면?

```
a, b = b, a
```

소괄호가 생략된 상태

```
tmp = a
a = b
b = tmp
```

1. tmp에 a값(10) 미리 보관
2. a에 b값(15) 저장
3. b에 미리 보관한 a값(10) 저장

**임시 변수; temporary의 의미*

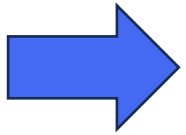
Unpacking 활용 (2): 입력 분할 input().split()

한 줄에 공백을 기준으로 여러 개의 값을 입력 받기 위함.

```
a, b, c = input().split()
```

사용자 입력

한 줄 입력



```
10 20 30
```

문자열 “10 20 30”과 동일

Unpacking 활용 (2): 입력 분할 input().split()

한 줄에 공백을 기준으로 여러 개의 값을 입력 받기 위함.

```
a, b, c = input().split()
```

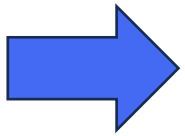
str.split(c) → List

문자열을 대상으로 문자열 c를 기준으로
문자열을 분할하여 각 분할된 값을 리스트로 묶어서 반환

**인자를 비워놓으면 Default로 공백(" ")을 기준으로 수행*

```
10 20 30
```

입력 받은 값
공백을 기준으로 분할

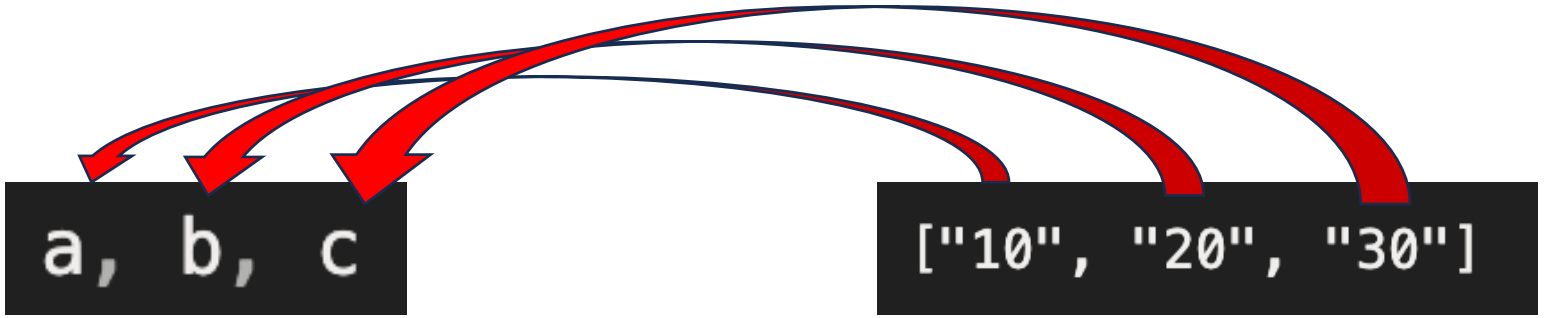


```
["10", "20", "30"]
```

Unpacking 활용 (2): 입력 분할 input().split()

한 줄에 공백을 기준으로 여러 개의 값을 입력 받기 위함.

```
a, b, c = input().split()
```



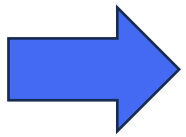
언패킹으로 리스트를 해체하여 각 원소를 변수에 할당

입력 분할의 확장: map() 함수

분할은 했지만, 실질적으로는 모두 **문자열 타입**이다! 일일이 다 숫자형으로 변환?...
입력 받은 값을 한 번에 숫자로 만들자! **map() 함수**

```
a, b, c = input().split()

print(type(a))
print(type(b))
print(type(c))
```



```
10 20 30
<class 'str'>
<class 'str'>
<class 'str'>
```

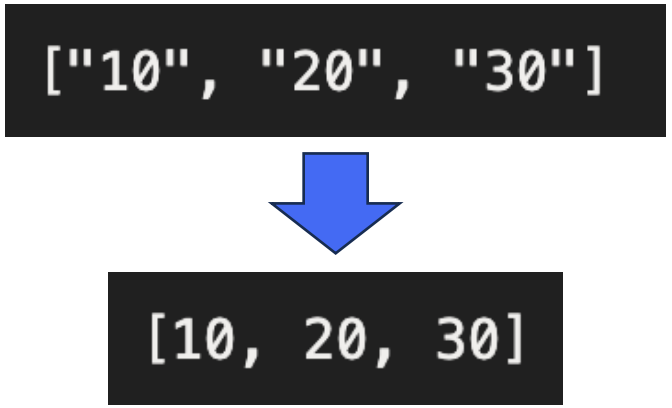
입력 분할의 확장: map() 함수

*map() 함수: 리스트의 각 원소에 변환 함수를 적용하여 변환된 새로운 리스트를 반환

map(변환 함수, 변환할 반복 가능한 자료형)

```
a, b, c = list(map(int, input().split()))
```

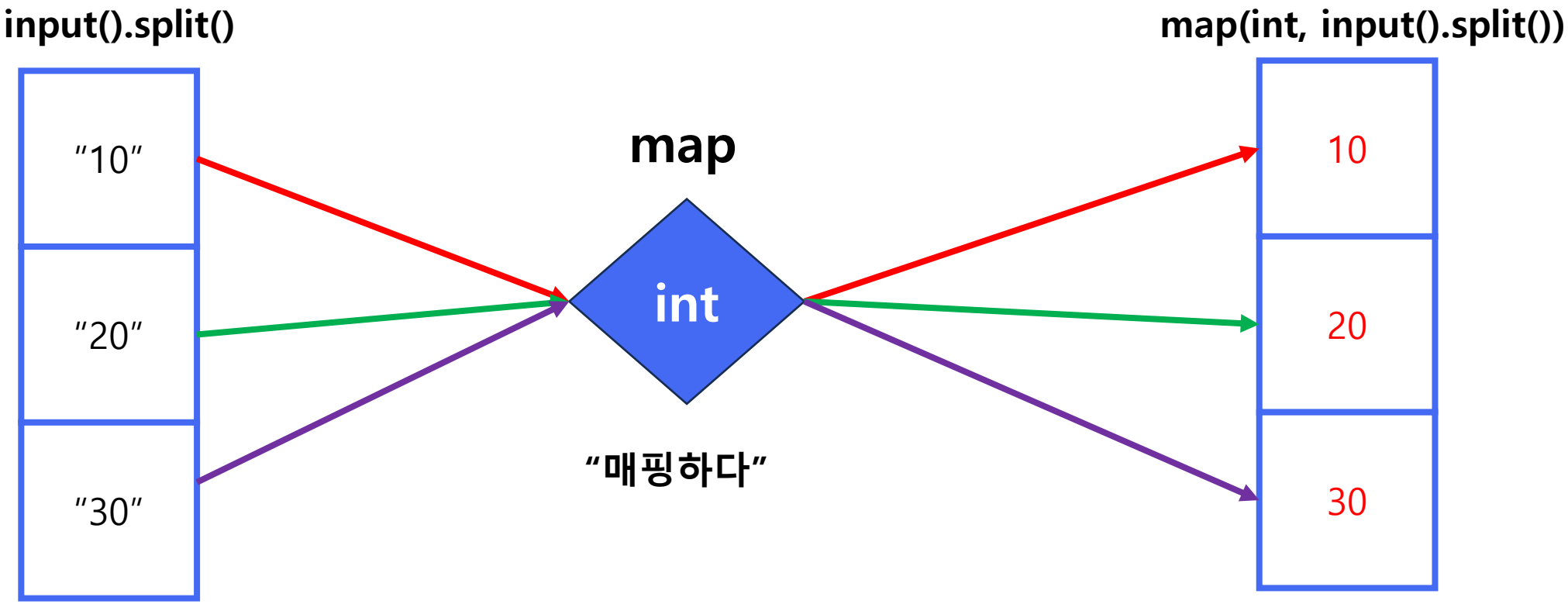
input().split()의 결과 리스트를 대상으로 각 원소마다 정수로 변환하여 새로운 리스트 생성



*map() 함수 자체의 결과는 리스트는 아니기 때문에 list() 함수로 형 변환해야 리스트 형태로 받을 수 있다.

입력 분할의 확장: map() 함수

변환 함수를 거쳐 기존의 값을 새로운 값으로 변환

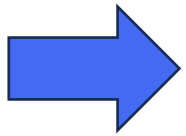


입력 분할의 확장: map() 함수

map() 함수: 리스트의 각 원소에 변환 함수를 적용하여 변환된 새로운 리스트를 반환
그리고 언패킹을 통해 변환된 리스트의 각 원소를 변수에 할당

```
a, b, c = list(map(int, input().split()))

print(type(a))
print(type(b))
print(type(c))
```



```
10 20 30
<class 'int'>
<class 'int'>
<class 'int'>
```

한 줄에 걸쳐 여러 숫자를 입력 받을 때 이와 같은 방식을 사용

Dictionary 자료형

딕셔너리(Dictionary) 자료형의 필요성

묶음이 있는 데이터를 봤을 때, 각 값 의미?

각 값이 정확히 무엇을 의미하는지 바로 파악하기 어려운 문제가 있다.

```
person = ("Gildong", 25, "서울시 강남구")
```

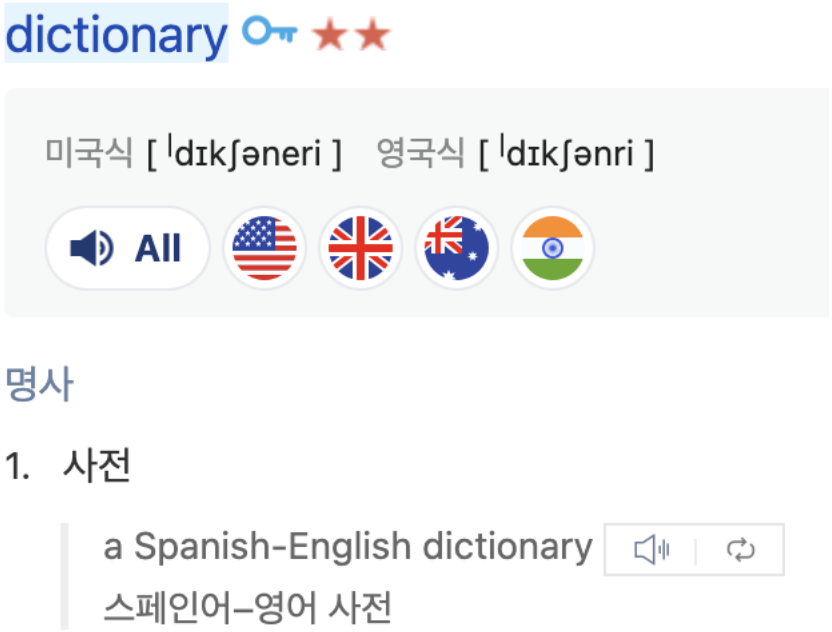
```
product = ("P001", "OO상품", 1400, 25, 0.0)
```

? ? ? ? ?

딕셔너리(Dictionary) 자료형

Dictionary = 사전! 뜻 그대로의 자료형

사전에서 단어와 뜻이 하나의 쌍인 것처럼 쌍을 이루는 자료형



성 명	
이 메 일	
휴 대 폰	
주 소	

딕셔너리 자료형

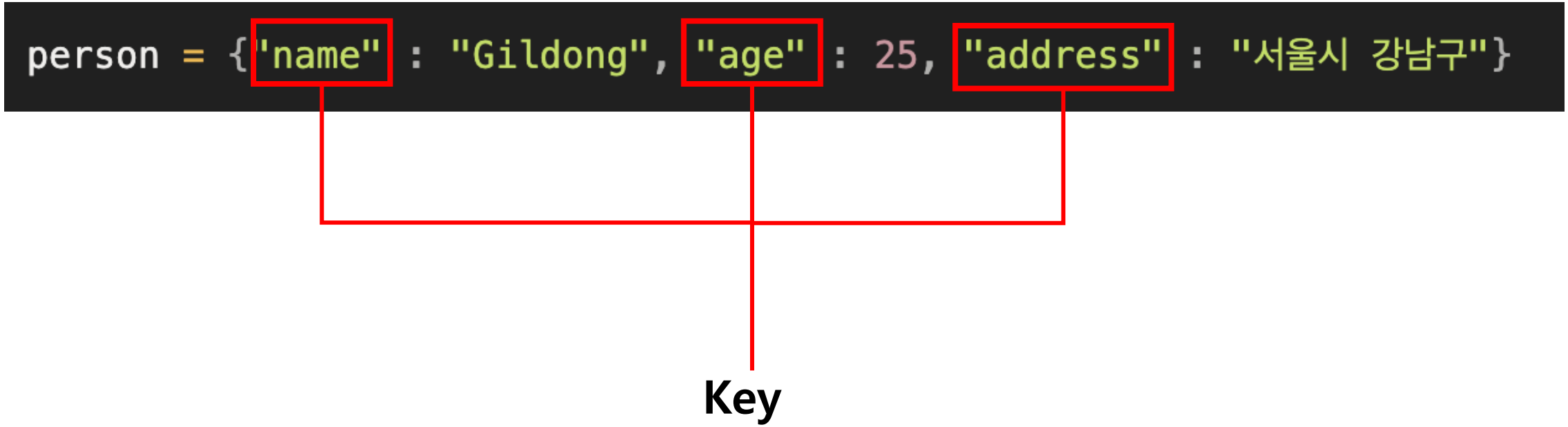
조금 더 쉽게 말하면, 값에 이름(이름표)을 붙인 것으로, 키:값 쌍들의 순서가 없는 집합

중괄호와 콜론(:)을 사용하여 표현. key : value가 하나의 쌍이며, 각 쌍은 콤마(,)로 구분

```
person = {"name" : "Gildong", "age" : 25, "address" : "서울시 강남구"}
```

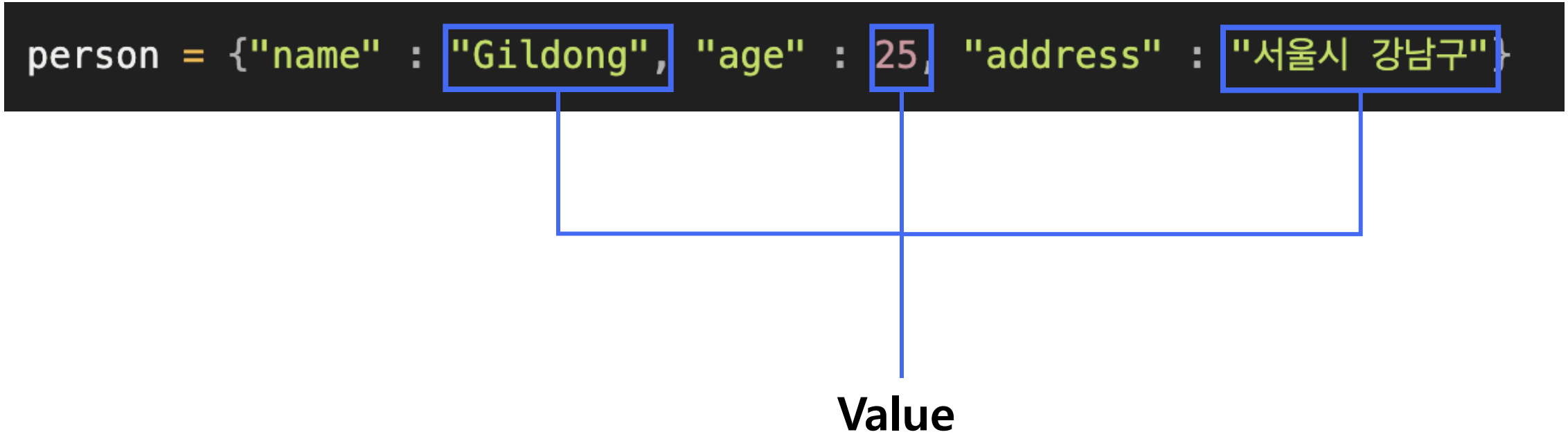
딕셔너리 자료형의 구성요소: Key

열쇠의 역할을 하는 Key를 알면 Value를 알 수 있다.



딕셔너리 자료형의 구성요소: Value

열쇠의 역할을 하는 Key에 의해 꺼내진 실제 값



딕셔너리 자료형 다루기 (1) : Key를 통해 Value 꺼내기(자료 읽기)

Key를 기반으로 Value 꺼내기 (리스트의 인덱싱과 유사! 하지만 값으로 접근!)

딕셔너리[Key]

```
person = {"name" : "Gildong", "age" : 25, "address" : "서울시 강남구"}

print(person["name"])
print(person["age"])
print(person["address"])
```

```
Gildong
25
서울시 강남구
```

딕셔너리 자료형 다루기 (2) : 자료 추가하기

기존에 없던 Key에 대해 접근 후, 새로운 Value 할당

딕셔너리[새로운Key] = Value

```
person = {"name" : "Gildong", "age" : 25, "address" : "서울시 강남구"}

person["gender"] = "M"
print(person)
```

새로운 "gender"키에 대해 "M" 할당

```
{'name': 'Gildong', 'age': 25, 'address': '서울시 강남구', 'gender': 'M'}
```

딕셔너리 자료형 다루기 (3) : 자료 수정하기

이미 존재하는 Key를 대상으로, 변경할 Value를 할당

딕셔너리[기존Key] = Value

```
person = {"name" : "Gildong", "age" : 25, "address" : "서울시 강남구"}

person["address"] = "서울시 마포구"
print(person)
```

기존의 “address”키에 대해 “서울시 마포구” 로 수정

```
{'name': 'Gildong', 'age': 25, 'address': '서울시 마포구'}
```

딕셔너리 자료형 다루기 (4) : 자료 삭제하기

삭제하고자 하는 대상 Key에 대해 del 키워드로 삭제

del 딕셔너리[Key]

```
person = {"name" : "Gildong", "age" : 25, "address" : "서울시 강남구"}

del person["age"]
print(person)
```

"age" 키에 대해 key:value 자료 삭제

```
{'name': 'Gildong', 'address': '서울시 마포구'}
```

딕셔너리 자료형 다루기 (5) : key:value의 포함 여부

in 연산자를 활용하여, 해당 Key가 딕셔너리에 있는지 검사

Key in 딕셔너리

```
person = {"name" : "Gildong", "age" : 25}

if "age" in person:
    person["age"] = 30
```

딕셔너리(Dictionary) 자료형

Example

```
level = {'low' : 1, 'medium' : 5}
```

(1) 딕셔너리 level에서 'medium'의 값을 가져오기

(2) 딕셔너리 level에서 'low' 키가 있는지 확인하기

(3) 딕셔너리 level에 {'high' : 10} 쌍을 추가하기

(4) 딕셔너리 level에서 'low' 키를 삭제하기

딕셔너리(Dictionary) 자료형

Example

```
level = {'low' : 1, 'medium' : 5}
```

(1) 딕셔너리 level에서 'medium'의 값을 가져오기

```
level['medium']
```

(2) 딕셔너리 level에서 'low' 키가 있는지 확인하기

```
'low' in level
```

(3) 딕셔너리 level에 {'high' : 10} 쌍을 추가하기

```
level['high'] = 10
```

(4) 딕셔너리 level에서 'low' 키를 삭제하기

```
del level['low']
```

딕셔너리(Dictionary) 자료형

딕셔너리 자료형의 특징: Key는 Immutable한 자료형으로!

딕셔너리의 키는 변할 수 없는 자료형(숫자, 문자열, 투플)으로 설정해야 한다.

- **리스트 같은 자료형으로 Key를 사용할 수 없다.**
- **Key 값이 바뀌면 짝꿍인 Value를 찾지 못하기 때문!**

**key의 해시 값을 Hash 테이블 기반으로 보관하는데, key가 바뀌면 해시 값도 변경됨.*

```
dict_ex = {[1, 2, 3] : "Hello"}
```

```
TypeError: unhashable type: 'list'
```

```
dict_ex = {(1, 2, 3) : "Hello"}
```

```
{(1, 2, 3): 'Hello'}
```

딕셔너리(Dictionary)

데이터 모델링(Data Modeling)

현실 세계의 데이터를 구조화 (딕셔너리 = JSON: API, DB 교환의 표준)

```
student = {
    "id" : 1001,
    "name" : "홍길동",
    "age" : 21,
    "major" : "Computer Science"
}
```

한 student의 데이터를 표현

```
company = {
    "name": "ABC Corp",
    "departments": {
        "HR": {
            "manager": "이영희",
            "employees": ["홍길동", "김민수"]
        },
        "IT": {
            "manager": "박지훈",
            "employees": ["최수빈", "정우성", "이서연"]
        }
    }
}
```

한 회사의 데이터를 표현 (중첩 딕셔너리)

딕셔너리(Dictionary)의 활용

딕셔너리의 메서드: keys(), values(), items()

- keys() : 현재 딕셔너리의 모든 key들의 묶음을 반환.
- values() : 현재 딕셔너리의 모든 value들의 묶음을 반환.
- items() : 현재 딕셔너리의 key-value 쌍을 Tuple로 묶은 묶음을 반환.

```
capital_dict = {'Korea' : 'Seoul', 'Vietnam' : 'Hanoi', 'Japan' : 'Tokyo'}

print(capital_dict.keys())
print(capital_dict.values())
print(capital_dict.items())
```

```
dict_keys(['Korea', 'Vietnam', 'Japan'])
dict_values(['Seoul', 'Hanoi', 'Tokyo'])
dict_items([('Korea', 'Seoul'), ('Vietnam', 'Hanoi'), ('Japan', 'Tokyo')])
```

딕셔너리(Dictionary)의 활용

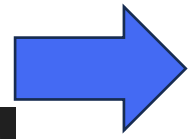
for문과 함께 활용: keys(), values()

딕셔너리의 원소(key-value 쌍)를 하나씩 순회하여 적용

```
capital_dict = {'Korea' : 'Seoul', 'Vietnam' : 'Hanoi', 'Japan' : 'Tokyo'}

for x in capital_dict:
    print(x)
```

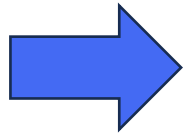
```
for x in capital_dict.keys():
    print(x)
```



```
Korea
Vietnam
Japan
```

```
capital_dict = {'Korea' : 'Seoul', 'Vietnam' : 'Hanoi', 'Japan' : 'Tokyo'}

for x in capital_dict.values():
    print(x)
```



```
Seoul
Hanoi
Tokyo
```

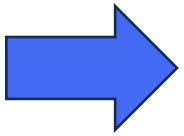
딕셔너리(Dictionary)의 활용

for문과 함께 활용: items()

언패킹을 활용하여 Key-Value 묶음 투플을 두 개의 변수로 할당

```
capital_dict = {'Korea' : 'Seoul', 'Vietnam' : 'Hanoi', 'Japan' : 'Tokyo'}

for x, y in capital_dict.items():
    print(x, y)
```



```
Korea Seoul
Vietnam Hanoi
Japan Tokyo
```

딕셔너리 활용: dict() 함수

딕셔너리 형 변환 함수로, 다중 원소로 구성된 리스트나 투플을 딕셔너리로 변환.

```
info1 = {'name' : 'Gildong', 'year' : 1999}
li = [('name', 'Gildong'), ('year', 1999)]

info2 = dict(li)
print(info2)
```

단, 내부 리스트(또는 투플)의 원소가 두 개여야 함

```
{'name': 'Gildong', 'year': 1999}
```

첫번째 원소가 Key, 두번째 원소가 Value

딕셔너리 활용: zip() 함수

여러 리스트나 투플을 동일한 인덱스끼리 하나의 투플로 묶어주는 역할

```
title = ['name', 'age', 'year']
values = ['John', 30, 1996]

print(zip(title, values))
print(list(zip(title, values)))
```

```
<zip object at 0x10d8a4b00>
[('name', 'John'), ('age', 30), ('year', 1996)]
```

딕셔너리 활용: zip() 함수의 확장

2개 뿐만 아니라, 그 이상의 개수도 묶기 가능

```
a = [1, 2, 3]
b = [10, 20, 30]
c = [100, 200, 300]
d = [1000, 2000, 3000]

print(list(zip(a, b, c, d)))
```

4개를 묶기!

```
[(1, 10, 100, 1000), (2, 20, 200, 2000), (3, 30, 300, 3000)]
```

딕셔너리(Dictionary)의 활용

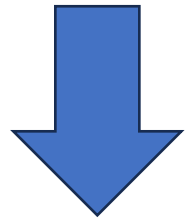
딕셔너리 활용: zip() 함수의 확장 - 만약 각각 길이가 다른 경우에는?

가장 짧은 리스트(또는 투플)의 길이에 맞춘다

```
a = [1, 2, 3]
b = [10, 20]
c = [100, 200, 300, 400]

print(list(zip(a, b, c)))
```

길이가 a=3, b=2, c=4인데 어떻게 묶일까?...



가장 짧은 b의 길이를 기준으로 묶는다

```
[(1, 10, 100), (2, 20, 200)]
```

딕셔너리 활용: zip() 함수와 dict() 함수

zip으로 묶은 결과를 딕셔너리로 변환: 첫번째 원소가 Key, 두번째 원소가 Value

```
title = ['name', 'age', 'year']
values = ['John', 30, 1996]

print(dict(zip(title, values)))
```

```
{'name': 'John', 'age': 30, 'year': 1996}
```

하나는 Key, 하나는 Value가 되는 구조이므로
2개의 리스트(또는 투플)로만 zip으로 묶어야 한다!

**만약에 여러 원소가 담긴 투플로 Key나 Value를 구성하고 싶다면 중첩 zip() 적용*

Exercise

**names와 scores라는 각각 빈 리스트를 정의하고,
5명의 학생의 이름과 점수를 입력 받아 각 리스트에 저장 후,
이 두 개의 리스트를 활용하여 딕셔너리를 생성하시오.**

```
이름을 입력하세요: Alice
점수를 입력하세요: 80
이름을 입력하세요: Bob
점수를 입력하세요: 90
이름을 입력하세요: Rosa
점수를 입력하세요: 70
이름을 입력하세요: David
점수를 입력하세요: 95
이름을 입력하세요: Maria
점수를 입력하세요: 85

{'Alice': 80, 'Bob': 90, 'Rosa': 70, 'David': 95, 'Maria': 85}
```

Exercise: Solution

names와 scores라는 각각 빈 리스트를 정의하고,
5명의 학생의 이름과 점수를 입력 받아 각 리스트에 저장 후,
이 두 개의 리스트를 활용하여 딕셔너리를 생성하시오.

```
names = []
scores = []

for i in range(5):
    n = input('이름을 입력하세요: ')
    s = int(input('점수를 입력하세요: '))
    names.append(n)
    scores.append(s)

result = dict(zip(names, scores))
print(result)
```

감사합니다!