문화 A0019

파이썬프로그래밍

김 태 완

kimtwan21@dongduk.ac.kr

오늘 배울 내용

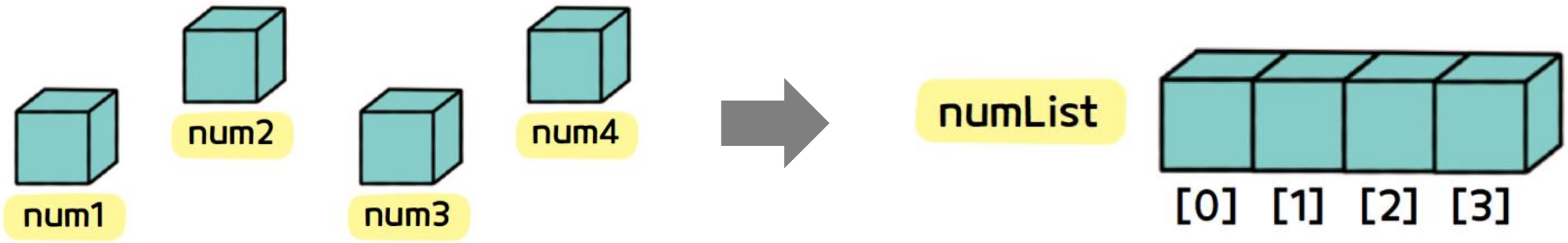
- 리스트

[1, 2, 3, 4, a]

- 리스트 값 접근
- 리스트에 새로운 값 삽입
- 리스트에 값 삭제
- 리스트에서 사용되는 함수

리스트

- 리스트 (list)
  - 하나씩 사용하던 변수를 붙여서 한 줄로 붙여 놓은 개념
  - 리스트는 개별 사물함 (변수)을 한 곳에 모아둔 전체 사물함 (리스트) 비유 가능
  - 아래 그림과 같이 리스트 이름 (numList)을 정하여 해당 리스트에 속한 다양한 변수를 한 번에 접근 가능
  - 각각의 데이터에는 번호(첨자)를 붙여서 접근함



리스트

- 리스트 생성
  - 대괄호 안에 값을 선언
    - 한 리스트 안에 다양한 변수의 type이 함께 있을 수 있음 : myList[0] = 1, myList[1] = “hello”, myList[2] = 3.15

```
리스트 이름 = [값1, 값2, 값3, …]
```

    - 다수의 데이터를 각각 변수로 선언하는 경우 vs 리스트를 사용하는 경우

❶ 각 변수 사용

```
num1, num2, num3, num4 = 10, 20, 30, 40
num1 사용
num2 사용
num3 사용
num4 사용
```

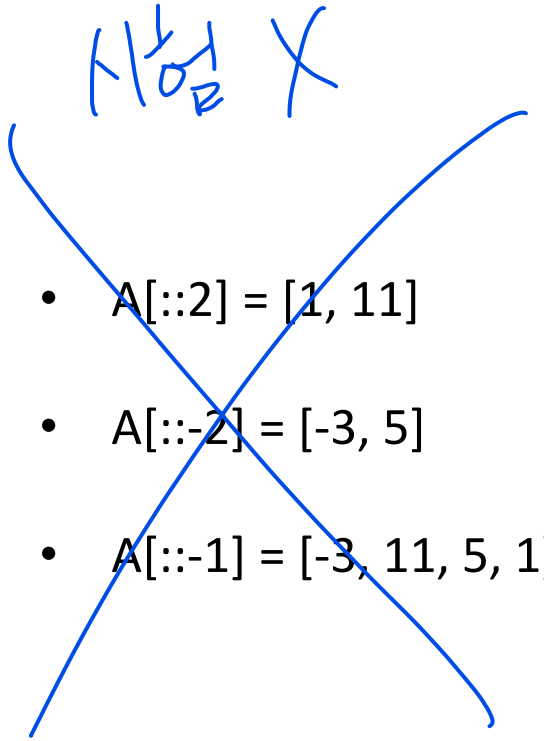
❷ 리스트 사용

```
numList = [10, 20, 30, 40]
numList[0] 사용
numList[1] 사용
numList[2] 사용
numList[3] 사용
```

리스트

- 리스트 값 접근

A = [1, 5, 11, -3]



- A[0] = 1
- A[1] = 5
- A[2] = 11
- A[3] = -3

- A[-1] = -3
- A[-2] = 11
- A[-3] = 5
- A[-4] = 1

- A[0:2] = [1, 5]
- A[2:4] = [11, -3]
- A[1:3] = [5, 11]
- A[1:] = [5, 11, -3]
- A[2:] = [11, -3]
- A[:2] = [1, 5]
- A[:3] = [1, 5, 11]

- A[::2] = [1, 11]
- A[::-2] = [-3, 5]
- A[::-1] = [-3, 11, 5, 1]

리스트

- 리스트끼리 덧셈과 곱셈

A = [1, 5, 11, -3]

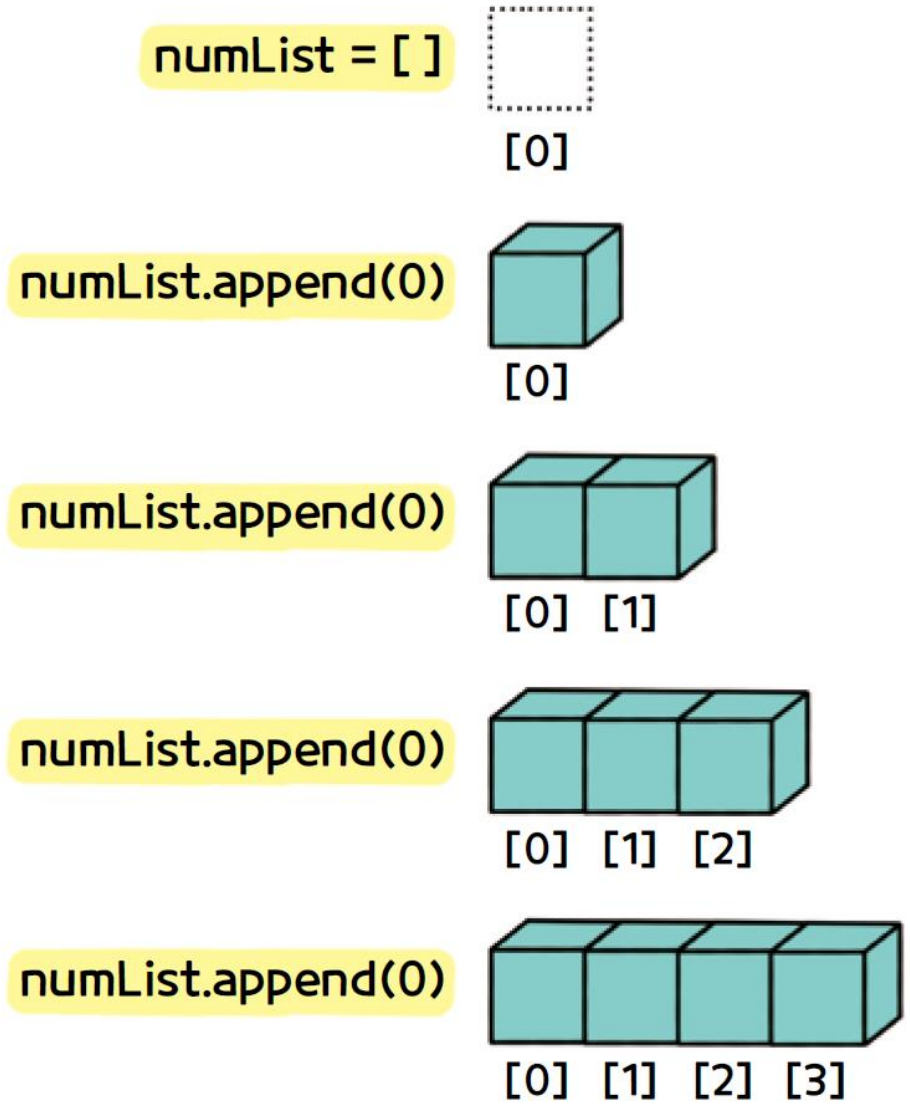
B = [‘data’, 0, ‘hi’, 0]

- A + B = [1, 5, 11, -3, ‘data’, 0, ‘hi’, 0]
- A*2 = [1, 5, 11, -3, 1, 5, 11, -3]
- B*3 = [‘data’, 0, ‘hi’, 0, ‘data’, 0, ‘hi’, 0, ‘data’, 0, ‘hi’, 0]

리스트

- 리스트 값 추가
  - append(value) 함수 사용 : 맨 뒤에 값 추가

```
>>> numList = [ ]
>>> numList.append(0)
>>> numList.append(0)
>>> numList.append(0)
>>> numList.append(0)
>>> print(numList)
[0, 0, 0, 0]
```



리스트

- 리스트 값 추가
  - insert 함수 사용 : 정해진 위치에 값 삽입

```
>>> numList = [1, 2, 3, 4]

>>> numList.insert(1, “hi”)

>>> print(numList)
[1, ‘hi’, 2, 3, 4]
```

리스트

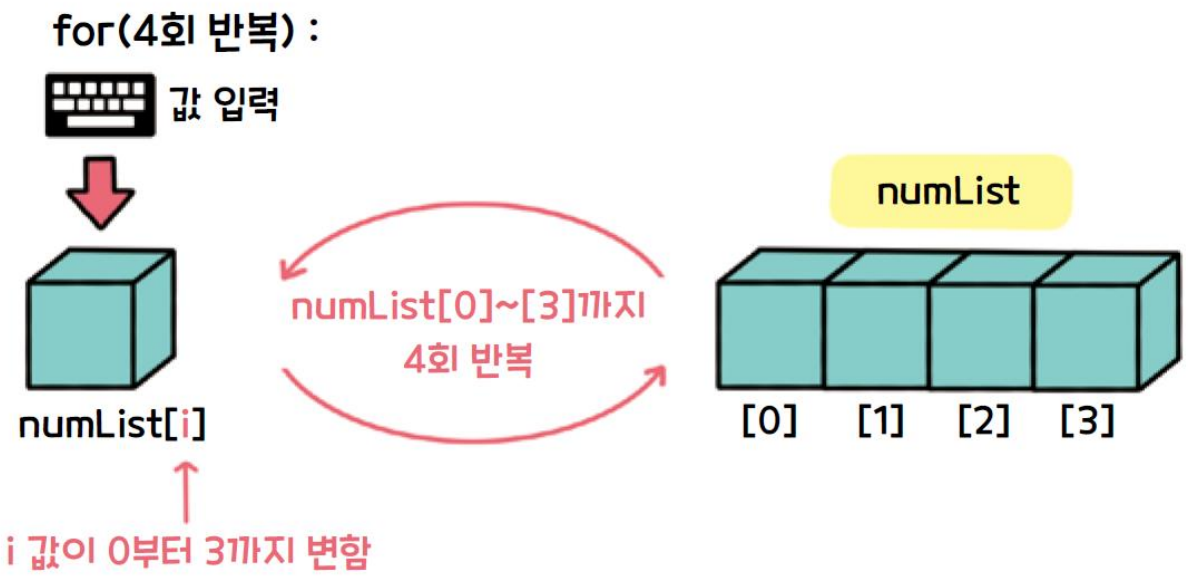
- 리스트 값 추가
  - 반복문 이용

```
>>> numList = []

>>> for i in range(0, 4) :
        numList.append(0)
```

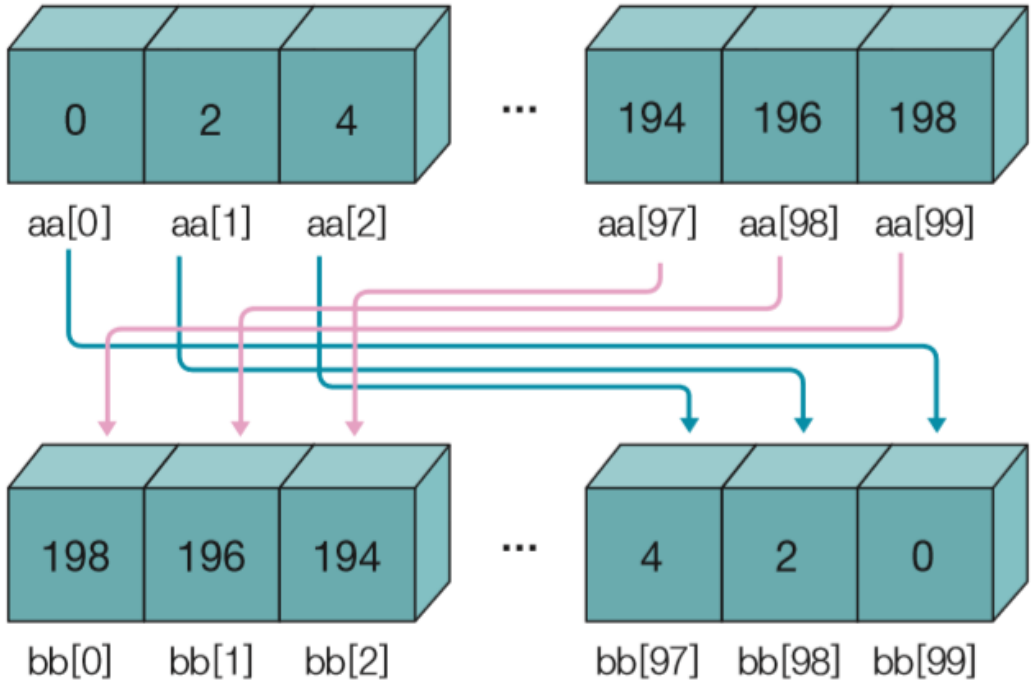
```
>>> numList = []

>>> for i in range(0, 4) :
        numList.append(i)
```



리스트

- 리스트 값 추가 예시
  - 리스트 100개인 aa 0, 2, 4, 8, …(2의 배수로 초기화 후 리스트 bb에 역순으로 넣는 과정)




```
1  aa = []
2  bb = []
3  value = 0
4
5  for i in range(0, 100) :
6      aa.append(value)
7      value += 2
8
9  for i in range(0, 100) :
10     bb.append(aa[99 - i])
11
12 print("bb[0]에는 %d이, bb[99]에는 %d이 입력됩니다." % (bb[0], bb[99]))
```

리스트

- 리스트 값 변경

A = [1, 5, 11, -3]

```
>>> A[0] = “python”

>>> print(A)
[‘python’, 5, 11, -3]
```

```
>>> A[1:2] = [7, -15]

>>> print(A)
[1, 7, -15, 11, -3]
```

```
>>> A[0] = [“python”, 9]

>>> print(A)
[[‘python’, 9], 5, 11, -3]
```

리스트

- 리스트 값 삭제

A = [1, 5, 11, -3]

```
>>> del(A[0])

>>> print(A)
[5, 11, -3]
```

```
>>> A[1:3] = [ ]

>>> print(A)
[1, -3]
```

```
>>> del(A)

>>> print(A)
NameError: name 'A' is not defined
```

```
>>> A = [ ]

>>> print(A)
[ ]
```

리스트

- 리스트 값 삭제
  - remove (지울 값) : 중복된 값을 지울 때, 모든 값을 지우지 않고 처음 만나는 한 개의 값만 삭제

A = [1, 5, 11, 5, -3]

```
>>> A.remove(11)

>>> print(A)
[1, 5, -3]
```

```
>>> A.remove(5)

>>> print(A)
[1, 11, 5, -3]
```

리스트

- 리스트 값 추출
  - pop() : 리스트 내 가장 뒤의 값을 뽑아내서 삭제함

A = [1, 5, 11, 5, -3]

```
>>> A.pop( )

>>> print(A)
[1, 5, 11, 5]
```

```
>>> print(A.pop( ))
-3

>>> print(A)
[1, 11, 5, -3]
```

리스트

- 리스트 조작 함수

함수	설명	사용법
append()	리스트 맨 뒤에 항목을 추가한다.	리스트명.append(값)
pop()	리스트 맨 뒤의 항목을 빼낸다(리스트에서 해당 항목이 삭제된다).	리스트명.pop()
sort()	리스트의 항목을 정렬한다.	리스트명.sort()
reverse()	리스트 항목의 순서를 역순으로 만든다.	리스트명.reverse()
index()	지정한 값을 찾아 해당 위치를 반환한다.	리스트명.index(찾을값)
insert()	지정된 위치에 값을 삽입한다.	리스트명.insert(위치, 값)
remove()	리스트에서 지정한 값을 삭제한다. 단 지정한 값이 여러 개면 첫 번째 값만 지운다.	리스트명.remove(지울값)
extend()	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 기능이 동일하다.	리스트명.extend(추가할리스트)
count()	리스트에서 해당 값의 개수를 센다.	리스트명.count(찾을값)
clear()	리스트의 내용을 모두 지운다.	리스트명.clear()
del()	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트명[위치])
len()	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트명)
copy()	리스트의 내용을 새로운 리스트에 복사한다.	새리스트=리스트명.copy()
sorted()	리스트의 항목을 정렬해서 새로운 리스트에 대입한다.	새리스트=sorted(리스트)

리스트

- 리스트 조작 함수 예
  - sort() : 리스트의 값을 정렬함
  - sort(reverse = True) : 내림차순으로 정렬함

A = [1, 5, 11, 5, -3]

```
>>> A.sort( )

>>> print(A)
[-3, 1, 5, 5, 11]
```

```
>>> A.sort(reverse=True)

>>> print(A)
[11, 5, 5, 1, -3]
```

```
>>> A.reverse( )

>>> print(A)
[-3, 5, 11, 5, 1]
```

리스트

- 리스트 조작 함수 예
  - count(찾을 값) : 찾을 값이 몇 개인지 개수를 카운트

A = [1, 5, 11, 5, -3]

```
>>> print(A.count(5))
2
```

  - copy ( ) : 리스트를 새로운 리스트로 복사

A = [1, 5, 11, 5, -3]

```
>>> B = A.copy( )

>>> print(B)
[1, 5, 11, 5, -3]
```

예시

- **예제 1 : greetings 리스트 값 순서대로 출력**

```
greetings = [‘안녕', '니하오', '하이', '곤니찌와', '올라', '싸와디캅', '봉쥬르']

# coding here #
```

- **실행 결과**

```
안녕
니하오
하이
곤니찌와
올라
싸와디캅
봉쥬르
```

예시

- 예제 2 : a 리스트 값 중에서 길이가 5인 값들만 출력

```
a = ['alpha', 'bravo', 'charlie', 'delta', 'echo', 'foxtrot', 'golf',
'cat', 'school' 'hotel', 'india’]

# coding here #
print(b)
```

- 실행 결과

```
['alpha', 'bravo', 'delta', 'india’]
```

튜플

- 튜플
  - 읽기 전용의 리스트이며, 소괄호로 생성
  - 리스트와 매우 비슷하지만 값을 읽을 수만 있으며, 수정하거나 새로 값을 추가할 수 없음
  - 튜플은 중복된 원소가 있을 수 있으며, 정해진 순서가 있어 순서가 다르면 서로 다른 튜플 : $(1,2,3) \neq (1,3,2)$
- 튜플의 생성
  - 튜플은 소괄호로 생성하지만, 괄호가 없어도 무방함

```
>>> Tup1 = (10, 20, 30)
>>> print(Tup1)
(10, 20, 30)
```

```
>>> Tup2 = 10, 20, 30
>>> print(Tup2)
(10, 20, 30)
```

  - 튜플의 항목이 한 개일 때 튜플 뒤에 콤마(,)를 붙여야 함

```
>>> Tup3 = (10)
>>> print(Tup1)
10
```

```
>>> Tup4 = 10
>>> print(Tup2)
10
```

```
>>> Tup5 = (10,)
>>> print(Tup3)
(10,)
```

```
>>> Tup6 = 10,
>>> print(Tup4)
(10,)
```

튜플

- 튜플은 읽기 전용
  - 다음의 세 가지 경우는 모두 오류

```
>>> Tup1.append(40)
>>> Tup1[0] = 40
>>> del(Tup1[0])
```

  - 튜플 자체를 통째로 삭제하려면 del(튜플이름) 함수를 사용

```
>>> del(Tup1)
```

  - 리스트에서 사용한 sort( ), reverse( ) 등의 함수도 사용 불가

튜플

- 튜플 값 읽기
  - 튜플이름[인덱스] : 특정 항목에 접근

```
>>> Tup1 = (10, 20, 30, 40)
>>> print(Tup1[0])
10
>>> print(Tup1[0] + Tup1[1] + Tup1[2])
60
>>> print(Tup1[1:3])
(20, 30)
>>> print(Tup1[1:])
(20, 30, 40)
>>> print(Tup1[:3])
(10, 20, 30)
```

튜플

- 튜플 덧셈과 곱셈

```
>>> Tup1 = (10, 20, 30, 40)
>>> Tup2 = ('A', 'B')

>>> print(Tup1 + Tup2)
(10, 20, 30, 40, 'A', 'B')

>>> print(Tup2 * 3)
('A', 'B', 'A', 'B', 'A', 'B')
```

- 튜플 → 리스트 → 튜플

```
>>> Tup1 = (10, 20, 30, 40)

>>> list1 = list(Tup1)
>>> list1.append(50)

>>> Tup2 = tuple(list1)

>>> print(Tup2)
(10, 20, 30, 40, 50)
```

딕셔너리

- 딕셔너리
  - 2개의 쌍이 하나로 묶이는 자료구조
    - 예시 : ‘apple:사과’ 처럼 의미 있는 두 값을 연결해 구성
  - 중괄호 { }로 묶어 구성, 키(Key)와 값(Value)의 쌍으로 구성

```
딕셔너리변수 = { 키1:값1 , 키2:값2, 키3:값3, ... }
```

- 딕셔너리 생성
  - 키와 값을 사용자가 지정하는 것이지 어떤 값을 반드시 사용해야 하는 규정은 없으며, 딕셔너리는 순서가 없음
    - 생성한 순서대로 딕셔너리가 구성되어 있다는 보장 없음
  - 예시 : 키(Key)가 1, 2, 3이, 값(Value)이 'a', 'b', 'c ' 인 딕셔너리

```
>>> myDict = {1:'a', 2:'b', 3:'c’}

>>> print(myDict)
{1: 'a', 2: 'b', 3: 'c'}
```

딕셔너리

- 딕셔너리 생성 예제
  - 딕셔너리는 여러 개의 정보를 하나의 변수로 표현할 때 유용하게 사용됨
    - 교수 김태완의 정보를 딕셔너리 구조로 생성하기

키 (key)	값 (value)
학번	20241234
이름	김태완
부서	데이터사이언스전공

```
>>> Profdict = {‘학번’: ‘20241234’ , '이름' : '김태완', '부서' : '데이터사이언스전공’}

>>> print(Profdict)
{‘학번’: ‘20241234’ , '이름' : '김태완', '부서' : '데이터사이언스전공’}
```

딕셔너리

- 딕셔너리 정보 추가
  - 정보를 추가할 때는 키와 값을 쌍으로 추가해야 함
  - '딕셔너리이름[키] = 값' 형식을 사용함
  - 만약 기존 딕셔너리에 동일한 키가 있을 경우 값이 변경

- **키는 중복되지 않고 유일함**
  - 값은 중복되어도 상관없음
  - 키 값이 중복일 경우 마지막에 있는 키가 적용

키 (key)	값 (value)
학번	20241234
이름	김태완
부서	데이터사이언스전공
나이	**39**
연락처	**'010-1234-5678'**

```
>>> Profdict['나이'] = 39
>>> Profdict['연락처'] = '010-1234-5678'

>>> print(Profdict)
{'학번': '20241234' , '이름' : '김태완', '부서': '데이터사이언스전공', '나이':39, '연락처': '010-1234-5678'}
```

딕셔너리

- 딕셔너리 정보 수정

키 (key)	값 (value)
학번	20241234
이름	**홍길동**
부서	데이터사이언스전공

```
>>> Profdict = {‘학번’: ‘20241234’ , '이름' : '김태완', '부서' : '데이터사이언스전공’}

>>> Profdict[‘이름’] = ‘홍길동’

>>> print(Profdict)
{‘학번’: ‘20241234’ , '이름' : '홍길동', '부서' : '데이터사이언스전공’}
```

딕셔너리

- 딕셔너리 키와 값 삭제
  - del(딕셔너리이름[키]) 함수를 사용

```
>>> Profdict = {‘학번’: ‘20241234’ , '이름' : '김태완', '부서' : '데이터사이언스전공’}

>>> del(Profdict[‘부서’])

>>> print(Profdict)
{‘학번’: ‘20241234’ , '이름' : '김태완'}
```

- 딕셔너리 값 읽기

```
>>> Profdict = {‘학번’: ‘20241234’ , '이름' : '김태완', '부서' : '데이터사이언스전공’}

>>> print(Profdict[‘학번’])
‘20241234’
>>> print(Profdict[‘부서’])
‘데이터사이언스전공’
```

딕셔너리

- 딕셔너리 함수
  - **keys( ) : 딕셔너리의 모든 키만 뽑아서 출력**

```
>>> Profdict = {‘학번’: ‘20241234’ , ' 이름 ' : ' 김태완', '부서' : ' 데이터사이언스전공’}

>>> print(Profdict.keys())
dict_keys([‘학번’, ‘이름’, ‘부서’])

>>> print(list(Profdict.keys()))
[‘학번’, ‘이름’, ‘부서’]
```

  - **get(키) : get 함수에 key 값을 이용하여 value 값에 접근 가능**

```
>>> Profdict = {‘학번’: ‘20241234’ , ' 이름 ' : ' 김태완', '부서' : ' 데이터사이언스전공’}

>>> print(Profdict.get(‘이름’))
‘김태완’
```

딕셔너리

- 딕셔너리 함수
  - **values( ) : 딕셔너리의 모든 값만 뽑아서 출력**
    - dict_values가 보기 싫으면 list(딕셔너리이름.values()) 함수를 사용

```
>>> Profdict = {‘학번’: ‘20241234’ , ' 이름 ' : ' 김태완', '부서' : ' 데이터사이언스전공’}

>>> print(Profdict.values())
dict_values([‘20241234’, ‘김태완’, ‘데이터사이언스전공’])

>>> print(list(Profdict.values()))
[‘20241234’, ‘김태완’, ‘데이터사이언스전공’]
```

  - **items( ) : 튜플 형태로 구하기**

```
>>> print(Profdict.items())
dict_items([(‘학번’,‘20241234’), (‘이름’,‘김태완’), (‘부서’,‘데이터사이언스전공’)])
```

딕셔너리

- 딕셔너리 함수
  - **in : 딕셔너리 안에 키가 있는지는 확인**

```
>>> Profdict = {‘학번’: ‘20241234’ , '이름' : '김태완', '부서' : '데이터사이언스전공’}

>>> print(‘학번’ in Profdict)
True

>>> print(‘연락처’ in Profdict)
False
```

딕셔너리

- 딕셔너리 함수
  - 반복문을 이용하여 딕셔너리에 저장된 키와 값 모두 출력 가능

```
>>> Profdict = {‘학번’: ‘20241234’ , '이름':'김태완', '부서':'데이터사이언스전공’}

>>> for key in Profdict.keys():
        print(key, Profdict[key])

>>> for k,v in Profdict.items():
        print(k, v)
```

예제

- **끝 입력 전까지 무한 반복문 생성**

```
foods = {"떡볶이":"김밥", "자장면":"단무지","라면":"파김치","치킨":"맥주
","삼겹살":"소주"}

while True:
# coding here #
```

- **실행 결과**

```
['떡볶이', '자장면', '라면', '치킨', '삼겹살']중 좋아하는 음식은? 자장면
자장면 궁합 음식은 단무지 입니다.
['떡볶이', '자장면', '라면', '치킨', '삼겹살']중 좋아하는 음식은? 라면
라면 궁합 음식은 파김치 입니다.
['떡볶이', '자장면', '라면', '치킨', '삼겹살']중 좋아하는 음식은? 치킨
치킨 궁합 음식은 맥주 입니다.
['떡볶이', '자장면', '라면', '치킨', '삼겹살']중 좋아하는 음식은? 끝
```

예제

- **튜플로 관리자 정보 처리**

```
admin_info = [‘admin’, ‘12345’, ‘data@dongduk.ac.kr']

# coding here #
```

- **실행 결과**

```
👉 실행 결과 1
관리자 아이디를 입력하세요 : rubato
관리자 비밀번호를 입력하세요 : 1111
아이디 또는 비밀번호가 잘못 입력되었습니다.

👉 실행 결과 2
관리자 아이디를 입력하세요 : admin
관리자 비밀번호를 입력하세요 : 12345
관리자입니다.
```

예제

- **딕셔너리로 성적 합계/평균 구하기**

```
scores = {'김예진': 90, '박영진': 95, '김소희': 84}
sum = 0

for key in scores:
    sum += 1
    print('%s : %d' % ( 2 , scores[key]))

avg = sum/len( 3 )

print('합계 : %d, 평균 : %.2f' % (sum, avg))
```

- **실행 결과**

```
👉 실행 결과
김예진 : 90
박영진 : 95
김소희 : 84
합계 : 269, 평균 : 89.67
```

감사합니다

kimtwan21@dongduk.ac.kr

김 태 완