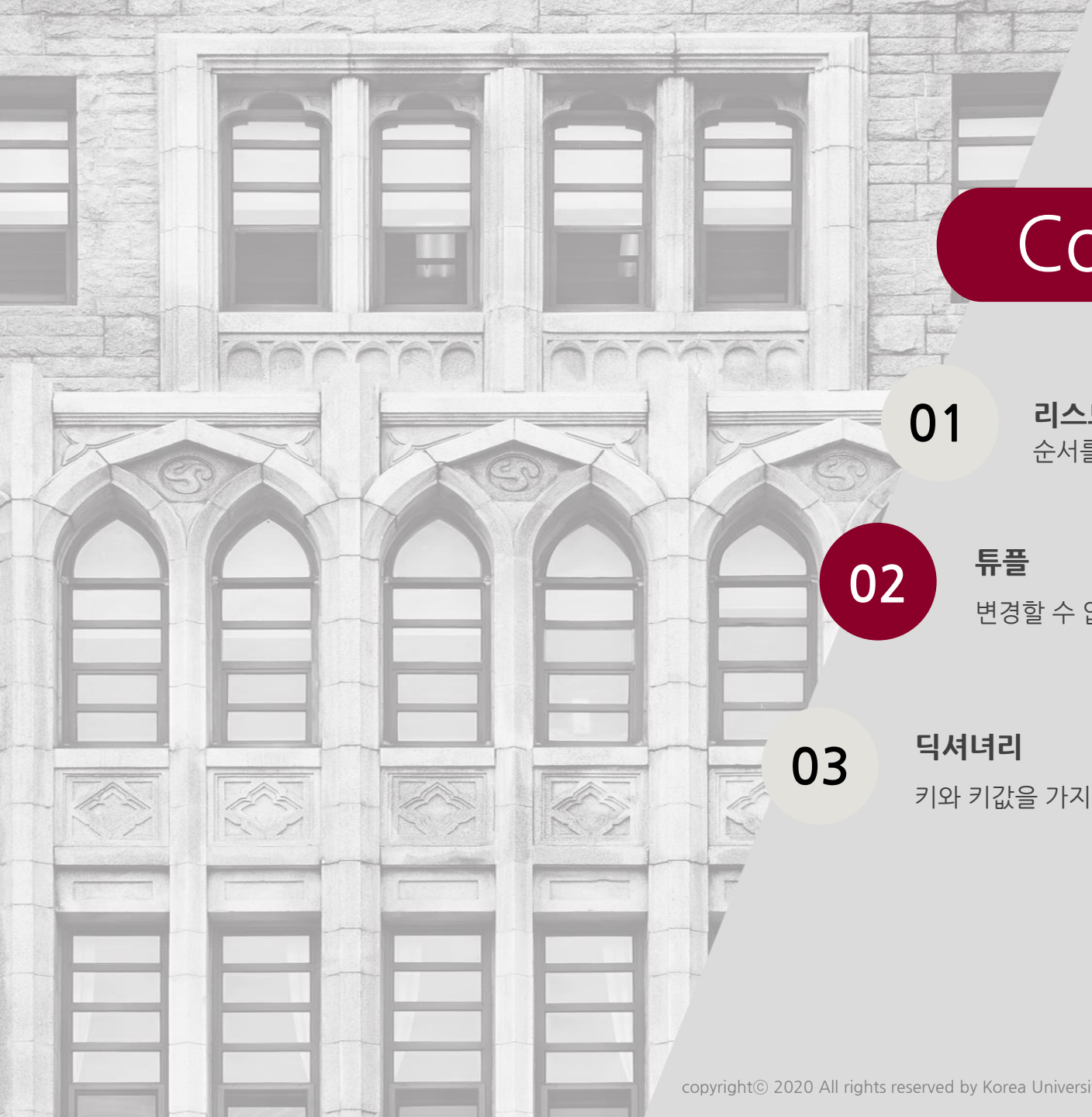
모두를 위한 파이썬 프로그래밍

6주 파이썬 데이터 타입
(List, Tuple, Dictionary)

생각해보기

- **파이썬에서의 다양한 데이터 타입**
  - 정수형
  - 실수형
  - 문자형
  - Bool형
  - List
  - Tuple
  - Dictionary



Contents

리스트(list) 1

리스트(List)의 이해

■ 리스트의 개념

- ✓ 하나의 변수에 여러 값을 할당하는 자료형
  - 파이썬에서는 여러 데이터를 하나의 변수에 할당하는 기법을 시퀀스 자료형
- ✓ 리스트는 [ ]로 표시하고, 리스트 안에 요소들은 콤마로 구분하여 나열
- ✓ 리스트의 요소 : 숫자, 문자 , 문자열 , 리스트 , 튜플(Tuple) , 사전, 함수 , 클래스 등
- ✓ 정수형이나 실수형 같은 다양한 자료형을 함께 포함할 수 있음.

```
e = [1, 2, ‘a’, ['한국', 'university']]
```

리스트(List)의 이해

■ 리스트의 표현

✓ Colors 변수에 3개의 값을 저장

```
colors = ['red', 'blue', 'yellow']
```

✓ 대괄호([ ])로 감싸 주고 각 요소값은 쉼표(,)로 구분

리스트명 = [요소1, 요소2, 요소3, ...]

✓ 다양한 리스트 표현

```
a = []
b = [1, 2, 3]
c = ['this', 'is', 'my', 'book']
d = [1, 2, 'korea', 'university']
e = [1, 2, ['korea', 'university']]
```

리스트(List)의 이해

리스트 생성, 값 변경, 출력

```
1   def under_bar():
2       print('_' * 15)
3
4
5   under_bar()  #    함수 실행
6   list1 = [1, 2, 3, 4, 5]
7   list2 = ['a', 'b', 'c']
8   list3 = [1, 'a', 'abc', [1, 2, 3, 4, 5], ['a', 'b', 'c']]
9   list1[0] = 6         # 6으로 정정
10  print(list1)         # [6, 2, 3, 4, 5]
11  list4 = [1, 2, under_bar]
12  list4[2]()
13  print(list3[3][1])  # 2
```

리스트(List)의 이해

■ 인덱싱(indexing)

- ✓ 문자열처럼 리스트에 있는 값에 접근하기 위해서는 인덱스 주소 사용
- ✓ 네비게이션 안내를 위한 음성파일 구성 예

```
nv_agent = 'school'
ng_voices = ['', '잠시후', '전방에서', '좌회전', '우회전', '직진',
             '유턴', '어린이 보호', '단속', '구역', '버스전용차선',
             '하십시오', '입니다', '과속에', '주정차 단속', '조심하십시오',
             '10m', '50m', '100m', '300m', '1km']

if nv_agent == 'go_left': # 좌회전 상황일 때 출력 예
    print(ng_voices[1], ng_voices[16], ng_voices[2], ng_voices[3], ng_voices[11])
elif nv_agent == 'left_turn':
    print(ng_voices[1], ng_voices[17], ng_voices[2], ng_voices[3], ng_voices[11])
elif nv_agent == 'u_turn':
    print(ng_voices[1], ng_voices[18], ng_voices[2], ng_voices[6], ng_voices[11])
elif nv_agent == 'school':
    print(ng_voices[1], ng_voices[7], ng_voices[9], ng_voices[12], ng_voices[15])
```

리스트(List)의 이해

■ 인덱싱(indexing)

- 문자열에서의 인덱스 번호와 동일하게 사용

```
cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
          0     1     2     3     4     5     6     7
```

- 문자열과 마찬가지로 리스트에는 인덱스를 마지막 값부터 시작하는 **reverse index** 사용 가능

```
cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
         -8    -7    -6    -5    -4    -3    -2    -1
```

리스트(List)의 이해

■ 슬라이싱(Slicing)

- ✓ 리스트의 인덱스를 사용하여 전체 리스트에서 일부를 잘라내어 반환
- ✓ 슬라이싱 기본 문법

변수명[시작 인덱스번호:마지막 인덱스번호]

```
1  cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
2  print(cities)
3  print(cities[0:6])
4  print(cities[6:])
5  print(cities[-8:])
6  print(cities[-8:2])
```

```
['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
['서울', '부산', '인천', '대구', '대전', '광주']
['울산', '수원']
['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
['서울', '부산']
```

리스트(List)의 이해

■ 슬라이싱 : 범위 선택 연산자 이용

- 인덱스를 따로 넣지 않고 print(cities[:])과 같이 콜론(:)을 넣으면 cities 변수의 모든 값을 다 반환한다.
- 범위선택연산자는 인덱스를 넘어서거나 입력하지 않더라도 자동으로 시작 인덱스와 마지막 인덱스로 지정된다.

```
1  city = 'seoul'
2  cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
3  print(city)
4  print(city[:])
5  print(city[-10:10])
6  print(cities)
7  print(cities[:])
8  print(cities[-10:10])
9  # print(city[5])    # IndexError: string index out of range
```

리스트(List)의 이해

■ 슬라이싱 : 범위 선택 연산자의 증가값(step)이용

- 슬라이싱에서는 시작 인덱스와 마지막 인덱스 외에 마지막 자리에 증가값을 넣을 수 있다.
- 증가(감소)값은 한 번에 건너뛰는 값을 의미한다

변수명[시작 인덱스번호:마지막 인덱스번호:증가치]

```
1  city = 'seoul'
2  cities = ['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
3  print(city)
4  print(city[::2])
5  print(city[::-1])
6  print(cities)
7  print(cities[0:6:3])
8  print(cities[::-1])
```

```
seoul
sol
luoes
['서울', '부산', '인천', '대구', '대전', '광주', '울산', '수원']
['서울', '대구']
['수원', '울산', '광주', '대전', '대구', '인천', '부산', '서울']
```

리스트(List)의 연산

덧셈 연산

- 덧셈 연산을 수행하더라도 기존 변수는 변화가 없다.
- 덧셈 연산의 결과는 다른 변수에 저장해 주어야 이용할 수 있다.

```
1  cities1 = ['서울', '김포', '부산']
2  cities2 = ['인천', '대구', '대전', '광주']
3  print(cities1 + cities2)    # 리스트 붙여서 출력
4  korean_cities = cities1 + cities2
5  print(korean_cities)        # 통합된 새로운 리스트
```

리스트(List)의 연산

곱셈 연산

- 리스트의 곱셈 연산은 기준 리스트에 n을 곱했을 때, 같은 리스트를 n배만큼 늘려 준다.

in 연산

- 특정 데이터가 포함되어 있는지를 확인하는 연산으로, 하나의 값이 해당 리스트에 들어 있는지 확인할 수 있다. 결과값은 True / False
- 조건문에서 자주 사용되는 연산자

```
1  cities = ['서울', '김포', '부산']
2  print(cities * 2)   # 리스트 2회 반복하여 출력
3  korean_cities = cities * 2
4  print(korean_cities)
5  print('김포' in korean_cities)
```

```
['서울', '김포', '부산', '서울', '김포', '부산']
['서울', '김포', '부산', '서울', '김포', '부산']
True
```

리스트(List) 관련 주요 함수

■ 리스트 추가 및 삭제 함수

함수	기능	비고
append()	- 새로운 값을 추가	cities.append('광주')
extend()	- 새로운 리스트를 추가 - 덧셈 효과	cities.extend(['용산','대구'])
insert()	- 지정한 인덱스에 추가 - 추가로 인한 밀림 발생	cities.insert(0, '김포')
remove()	- 특정 값을 찾아서 제거	cities.remove('서울')
del	- 인덱스 값으로 삭제	del cities[0]

리스트(List) 관련 함수

리스트 추가 함수 : append( )

- 새로운 값을 마지막 인덱스에 추가
- 리스트 내에 새로운 리스트를 추가할 수 있음

```
1  city = 'korea'
2  cities = ['서울', '부산', '인천', '대구', '대전']
3  cities[4] = '일산'    # 수정
4  print(cities)
5  #cities[5] = '김포'    # 추가 안됨
6  city += ' university'
7  cities.append('김포')
8  print(cities)
9  print(city)
```

리스트(List) 관련 함수

■ 리스트 추가 및 삭제 함수

- **extend( ) 함수 :** 새로운 리스트를 기존 리스트에 추가
- 리스트 + 연산자의 기능과 동일 효과
- 문자열을 추가 하면 분리되어 리스트에 추가

```
1  cities = ['서울', '부산', '인천', '대구', '대전']
2  cities = cities + ['고양', '일산']    # 2개 추가
3  print(cities)
4  cities = cities + ['고양']    # 1개 추가
5  print(cities)
6  cities.extend(['독도'])
7  print(cities)
8  cities.extend('대한민국')
9  print(cities)
```

리스트(List) 관련 함수

■ 리스트 추가 및 삭제 함수

- **insert( ) 함수 :** 기존 리스트의 i번째 인덱스에 새로운 값을 추가, i번째 인덱스를 기준으로 뒤쪽의 인덱스가 하나씩 밀림.

- **remove( ) 함수 :** 리스트 내의 특정 값을 삭제.

```
1  cities = ['서울', '부산', '인천', '대구', '대전']
2  print(cities)
3  cities.insert(1, '김포')
4  cities.insert(8, '광주')  # 인덱스 초과  --> 제일 뒤에 추가됨
5  print(cities)
6  cities.remove('인천')
7  print(cities)
8  del cities[0]  # 인덱스 번호로 삭제
9  print(cities)
```

다중 리스트(List)

■ 리스트의 다중 구성

- 리스트를 효율적으로 활용하기 위해 여러 개의 리스트를 하나의 변수에 할당하는 이중, 삼중 리스트를 사용할 수 있다.

이름	jane	allice	john
국어 점수	56	67	88
영어 점수	45	76	77

- 이중리스트에 인덱싱하여 값에 접근하기 위해서는 대괄호 2개를 사용한다.

```
1  kor_score = [56, 67, 88]
2  eng_score = [45, 76, 77]
3  total = [kor_score, eng_score]
4  print(kor_score)
5  print(eng_score)
6  print(total)          # [[56, 67, 88], [45, 76, 77]]
7  print(total[0][2])    # 88
```

다중 리스트(List)

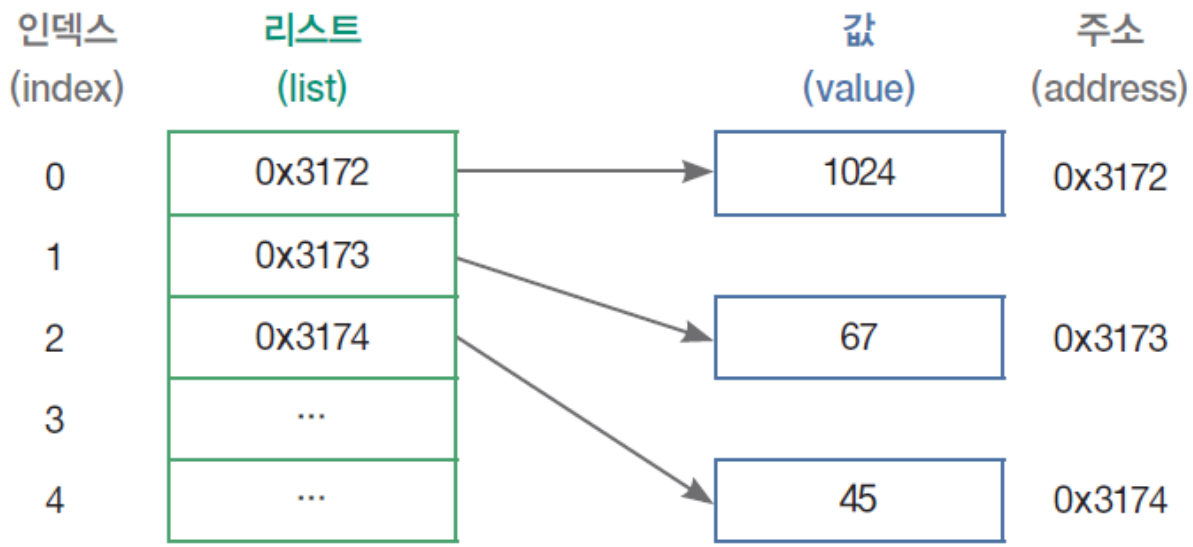
■ 다중 리스트 구성 시 유의사항

```
1   kor_score = [56, 67, 54]
2   eng_score = [45, 76, 43]
3   score_card_1 = kor_score + eng_score    # 새로운 리스트 생성
4   score_card_2 = [kor_score, eng_score]   # 기존 리스트의 틀만 변형 시켜 활용
5   print(kor_score)                        [56, 67, 54]
6   print(eng_score)                        [45, 76, 43]
7   print(score_card_1)                     [56, 67, 54, 45, 76, 43]
8   print(score_card_2)                     [[56, 67, 54], [45, 76, 43]]
9   kor_score[1] = 99
10  score_card_2[0][0] = 100
11  print(kor_score)                        [100, 99, 54]
12  print(eng_score)                        [45, 76, 43]
13  print(score_card_1)                     [56, 67, 54, 45, 76, 43]
14  print(score_card_2)                     [[100, 99, 54], [45, 76, 43]]
15  print(kor_score)                        [100, 99, 54]
```

다중 리스트(List)

■ 다중 리스트 구성 시 주의사항

- 기존 리스트를 이용하여 새로운 다중 리스트를 생성할 때 실제 값이 아닌, 원본 데이터가 저장된 위치의 메모리 주소(reference)를 저장(참조)하는 방식으로 만들어 진다.
  → 유닉스의 파일복사 방식(심볼릭 링크), Windows의 단축아이콘 생성 방식과 동일



튜플(Tuple) 2

튜플(tuple)

■ 튜플의 구성

- 리스트와 같은 개념이지만, 데이터를 변경할 수 없는 자료구조
- 대괄호 [ ]를 이용하는 리스트와 다르게 ()괄호를 사용
- 리스트에서 사용하는 연산, 인덱싱, 슬라이싱이 모두 동일

```
1  kor_score = (56, 67, 88)
2  eng_score = (45, 76, 77)
3  total = (kor_score, eng_score)
4  print(kor_score)
5  print(eng_score)
6  print(total)
7  print(total[0][2])  # 88
8  kor_score[0] = 100  # 'tuple' does not support item assignment
```

튜플(tuple)

■ 튜플의 활용

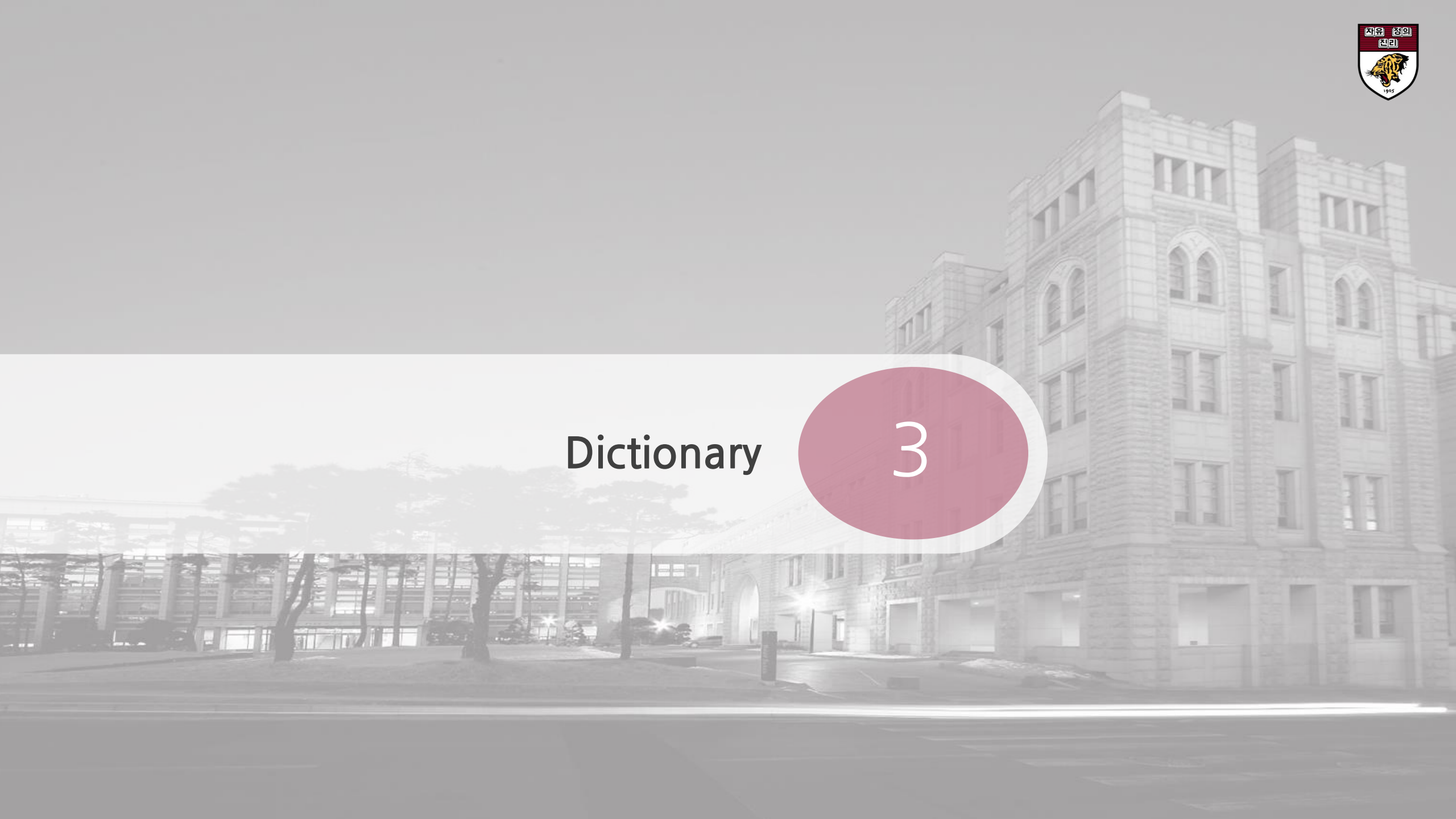
- **튜플은 언제 사용할까?**

  만들어 놓은 함수를 공개하면 다른 사람이 사용할 수 있다. 이때 사용자가 데이터를 읽어서 사용하는 것은 허용하나 포함된 데이터를 수정하는 것은 불가능하게 설정해야 하는 경우에 수정 금지 모드로 지정해 놓기 위해 사용한다.

- **그렇다면 수정되지 않아야 하는 데이터는 어떤 것이 있을까?**

  - 학번이나 이름, 우편번호, 주민등록번호, 주식 데이터, 학점 등

  - 프로그래머가 이러한 이해 없이 마음대로 값을 변경하려고 할 때, 튜플은 이를 방지하는 기능을 한다.



Dictionary 3

딕셔너리(dictionary)

딕셔너리 이해

- 전화번호부와 같이 키(key)와 값(value) 형태로 데이터를 저장하는 자료구조이다

.

학번	이름	생년월일	전화번호
20190001	홍길동	1992-01-02	010-1111-2222
20190002	이순신	1987-07-13	019-2222-2222
20190003	장보고	1999-05-22	011-3333-3333

- 파이썬에서 딕셔너리의 선언은 중괄호 {}를 사용하여 키와 값의 쌍으로 구성한다.

  딕셔너리 변수 = {키1:값1, 키2:값2, 키3:값3, . . .}

딕셔너리(dictionary)

딕셔너리 이해

이름(키)	전화번호(값)
홍길동	010-1111-2222
이순신	019-2222-2222
장보고	011-3333-3333

- tel_info라는 변수를 먼저 선언한 후, 해당 변수에 {키:값} 형태로 값을 입력

```
cellphone_info = {'홍길동': '010-1111-2222',
                  '이순신': '019-2222-2222',
                  '장보고': '011-3333-3333'}
```

딕셔너리(dictionary)

딕셔너리 이해

- **해당 변수에서 특정 값을 확인**

```
cellphone_info = {'홍길동': '010-1111-2222',
                  '이순신': '019-2222-2222',
                  '장보고': '011-3333-3333'}
print(cellphone_info['홍길동']) # 010-1111-2222
```

- **값의 수정과 데이터 추가**

```
cellphone_info['홍길동'] = '010-1234-5678' # 값 수정
print(cellphone_info['홍길동']) # 010-1234-5678
cellphone_info['유길상'] = '000-0000-0000' # 데이터 추가
print(cellphone_info)
```

딕셔너리(dictionary)

딕셔너리 함수

- 파이썬에서는 딕셔너리를 쉽게 사용할 수 있도록 다양한 함수를 제공한다.

```
print(cellphone_info.values())   # 값만 확인
print(cellphone_info.keys())     # 키만 확인
```

```
dict_values(['010-1111-2222', '019-2222-2222', '011-3333-3333'])
dict_keys(['홍길동', '이순신', '장보고'])
```

딕셔너리(dictionary)

딕셔너리 함수

- 키-값 쌍을 모두 보여 주기 위해서는 items( ) 함수를 사용

```
print(cellphone_info)    # 딕셔너리 출력
print(cellphone_info.items())    # 키:값 모두 확인
```

```
{'홍길동': '010-1111-2222', '이순신': '019-2222-2222', '장보고': '011-3333-3333'}
dict_items([('홍길동', '010-1111-2222'), ('이순신', '019-2222-2222'), ('장보고', '011-3333-3333')])
```

- if문을 사용하여 특정 키나 값이 해당 변수에 포함되어 있는지 확인

```
print('장보고' in cellphone_info.keys())    # True
print('011-3333-3333' in cellphone_info.values())    # True
```

딕셔너리(dictionary)

연습문제

- 학번을 키로 하여 각각의 키 값을 리스트 타입의 딕셔너리로 생성해 보자
- 딕셔너리를 출력해 보자
- 딕셔너리 값만 확인 해보자
- 딕셔너리 키만 확인 해 보자
- 딕셔너리 키와 값을 모두 확인해 보자

학번	이름	생년월일	전화번호
20190001	홍길동	1992-01-02	010-1111-2222
20190002	이순신	1987-07-13	019-2222-2222
20190003	장보고	1999-05-22	011-3333-3333

감사합니다.