Basic Python: Part I

Date	@2022년 8월 4일 오전 9:30
Code: Lecture	C2-004_Lecture_01.ipynb
Code: Practice	C2-004_Practice_01.ipynb
Lecture Note	C2-004_Lecture 01-basic 1.pdf
No.	1

▼ 1. 자료형(기본)

자료형: 자료의 형태(숫자, 문자열, True/False, 등)

▼ 1-1) 숫자 자료형

숫자 자료형: 정수(-8, +8), 실수(8.04, 3.14) 등 숫자 형태의 자료

```
# 정수 표현
print(8)  #  8 출력
print(-8) # -8 출력
```

```
# 실수 표현
print(8.04) # 8.04 출력
print(20220804) # 20220804 출력
```

```
# 간단한 연산
print(8+4) # 12 출력
print(8*4) # 32 출력 (곱셈연산자는 '*')
print(22*(8+4)) # 264 출력
```

▼ 1-2) 문자열 자료형

문자열 자료형: 한글, 영어 등 문자 형태의 자료

- Python 에서 문자열을 사용하기 위해서는 큰 따옴표(””) 또는 작은 따옴표(’) 필요

```
# 큰 따옴표 사용
print("중앙대학교") # 중앙대학교 출력
print("군 장병 AI.SW 역량강화사업") # 군 장병 AI.SW 역량강화사업 출력
```

```
# 작은 따옴표 사용
print('입문강의') # 입문강의 출력
print('Python/R 기초') # Python/R 기초 출력
```

```
# 간단한 응용
print('굿굿굿굿굿') # 굿굿굿굿굿 출력
print("굿"*5) # 굿굿굿굿굿 출력
```

▼ 1-3) Boolean 자료형

Boolean 자료형: 참(True) 또는 거짓(False) 형태의 자료

- Boolean 자료형은 True와 False 2가지 값만을 갖는다.

```
# Boolean 출력
print(True)  # True 출력
print(False) # False 출력
```

```
# 4가 8보다 크다: 참? 거짓?
print(4 > 8) # False 출력

# 4가 8보다 작다: 참? 거짓?
print(4 < 8) # True 출력
```

```
# "not" Boolean 출력
print(not True)  # False 출력
print(not False) # True 출력
```

```
# 4가 8보다 크지 않다: 참? 거짓?
print(not (4 > 8)) # True 출력

# 4가 8보다 작지 않다: 참? 거짓?
print(not (4 < 8)) # False 출력
```

▼ 1-4) 변수

변수: 자료형의 값을 저장하는 공간

```
# 군 장병 AI/SW 역량강화사업 입문과정 강사를 소개합니다.
print("김학구 교수는 Pytyhon/R기초 강의를 맡았습니다.")
print("김학구 교수는 2021년 2학기부터 중앙대학교 첨단영상대학원에 근무 중입니다.")
```

```
# 다른 과목을 맡은 강사를 소개할 경우:
print("최종원 교수는 패턴인식 강의를 맡았습니다.")
print("최종원 교수는 2020년 1학기부터 중앙대학교 AI대학원에 근무 중입니다.")
```

```
# 변수를 활용하는 방법
name = "김학구"
lecture = "Python/R기초"
year = 2021
semester = 2
department = "첨단영상대학원"

print(name + " 교수는 " + lecture + " 강의를 맡았습니다.")
print(name + " 교수는 " + str(year) + "년 " + str(semester) + "학기부터 " + "중앙대학교 " + department + "에 근무 중입니다.")
```

```
# 변수를 활용하는 방법
name = "최종원"
lecture = "패턴인식"
year = 2020
semester = 1
department = "AI대학원"

print(name + " 교수는 " + lecture + " 강의를 맡았습니다.")
print(name + " 교수는 " + str(year) + "년 " + str(semester) + "학기부터 " + "중앙대학교 " + department + "에 근무 중입니다.")
```

- 변수 사용 형태 1: (문자열+변수+문자열)
- 변수 사용 형태 2: (문자열, 변수, 문자열)
  - str() 형변환 사용할 필요 없음
  - 쉼표(,)를 이용하면 값과 값 사이에 공백이 한칸 포함됨

```
# 변수 사용 형태 1: (+)를 사용한 출력
print(name + " 교수는 " + lecture + " 강의를 맡았습니다.")
print(name + " 교수는 " + str(year) + "년 " + str(semester) + "학기부터 " + "중앙대학교 " + department + "에 근무 중입니다.")

# 변수 사용 형태 2: (,)를 사용한 출력
print(name, "교수는", lecture, "강의를 맡았습니다.")
print(name, "교수는", year, "년", semester, "학기부터", "중앙대학교", department, "에 근무 중입니다.")
```

▼ 1-5) 주석

주석: 프로그램의 실행에 영향을 미치지 않는 내용 지정(예: 코드 설명 등)

- 한 줄 주석(#): '#' 이후에 오는 내용은 모두 주석 처리
- 여러 줄 주석(Ctrl+/): 영역을 블록 지정 후, 단축키 "Ctrl+/"을 누르면 해당 영역 주석 처리

```
# 군 장병 AI/SW 역량강화사업 입문과정 강사를 소개합니다.
# print("김학구 교수는 Pytyhon/R기초 강의를 맡았습니다.")
# print("김학구 교수는 2021년 2학기부터 중앙대학교 첨단영상대학원에 근무 중입니다.")
```

▼ 2. 연산

▼ 2-1) 연산자

연산자	의미	예제
+	더하기	8 + 4 = 12
-	빼기	8 - 4 = 4
*	곱하기	8 * 4 = 32
/	나누기	8 / 4 = 2.0

```
# 연산자를 이용한 사칙연산
print(8 + 4) # 12
print(8 - 4) # 4
print(8 * 4) # 32
print(8 / 4) # 2.0
```

연산자	의미	예제
**	제곱	2 ** 3 = 8
%	나머지	5 % 3 = 2
//	몫	5 // 3 = 1

```
# 연산자를 이용한 다양한 연산
print(8 ** 4) # 8의 4제곱 = 4096
print(8 % 4) # 8을 4로 나눈 나머지 = 0
print(8 % 5) # 8을 5로 나눈 나머지 = 3
print(8 // 4) # 8을 4로 나눈 몫 = 2
print(8 // 5) # 8을 5로 나눈 몫 = 1
```

연산자	의미	예제
>	~보다 크다	8 > 4
>=	~보다 크거나 같다	4 >= 8
<	~보다 작다	8 < 4
<=	~보다 작거나 같다	8 <= 8

```
# 등호와 부등호를 이용한 크기 비교 연산
# 수식이 참이면 True, 거짓이면 False 출력
print(8 > 4)  # True
print(4 >= 8) # False
print(8 < 4)  # False
print(8 <= 8) # True
```

연산자	의미	예제
==	좌항과 우항이 같다	3 == 3
!=	좌항과 우항이 다르다	1 != 3

```
# 좌항과 우항이 같은 지 다른 지 비교 연산

# ==: 같으면 True, 다르면 False 출력
print(8 == 8) # True
print(8 == 4) # False
```

```
print(8 + 4 == 12) # True

# !=: 다르면 True, 같으면 False 출력
print(8 != 4) # True
print(8 - 4 != 4) # False
```

연산자	의미	예제
and	두 항이 모두 참이면 참이다.	(8 > 0) and (4 > 5)
or	두 항 중 하나라도 참이면 참이다.	(8 > 0) or (4 > 5)
not	~의 반대	not(8 != 4)

```
# 좌항과 우항이 모두 참인가?
print((8 > 0) and (4 > 5)) # False

# 좌항과 우항 중 하나라도 참인가?
print((8 > 0) or (4 > 5)) # True

# 좌항과 우항이 다른 지 비교한 결과의 반대는?
print(not(8 != 4)) # False
```

```
# 연속된 수식에 대한 비교 연산
print(8 > 4> 2) # (8 > 4): True and (4 > 2): True  -> True
print(8 > 4> 5) # (8 > 4): True but (4 > 5): False -> False
```

▼ 2-2) 간단한 수식

```
print(2 + 8 * 4) # 34
print((2 + 8) * 4) # 40
```

```
# 변수 사용1
number = 2 + 8 * 4
print(number) # 34
```

```
# 변수 사용2
number = number + 4
print(number) # 38
```

```
# 변수 사용3
number += 4
print(number) # 42
```

```
# 변수를 이용한 다양한 수식
number += 8
print(number) # 50

number -= 10
print(number) # 40

number /= 4
print(number) # 10

number *= 2
print(number) # 20

number %= 5
print(number) # 0
```

▼ 2-3) 숫자처리 함수

- Python 내장 함수를 이용한 숫자 처리

함수 이름	의미	예제

함수 이름	의미	예제
abs	절대값	abs(-8)
pow	제곱	pow(8, 4)
max	가장 큰 값	max(8, 4)
min	가장 작은 값	min(8, 4)
round	반올림	round(8.4)

```
# Python 내장 함수를 이용한 숫자 처리

print(abs(-8)) # -8 의 절대값 = 8
print(pow(8, 4)) # 8의 4제곱 = 4096
print(max(8, 4)) # 8 와 4 중 큰 값 = 8
print(min(8, 4)) # 8 와 4 중 작은 값 = 4
print(round(8.4)) # 8.4 의 반올림 = 8
print(round(4.8)) # 4.8 의 반올림 = 5
```

- math 모듈을 이용한 숫자 처리

  [모듈 사용 방법 1]

  : **from** 모듈이름 **import** 사용할 기능

함수 이름	의미	예제
floor	내림	floor(4.99)
ceil	올림	ceil(3.14)
sqrt	제곱근	sqrt(14)

```
# math 모듈을 이용한 숫자 처리 1

from math import * # math 모듈 내 모든 기능 사용

print(floor(4.8)) # 4.8 의 내림 = 4
print(ceil(8.4)) # 8.4 의 올림 = 9
print(sqrt(4)) # 4 의 제곱근 = 2.0
```

[모듈 사용 방법 2]

: **import** 모듈이름

```
# math 모듈을 이용한 숫자 처리 2

import math

print(math.floor(4.8)) # 4.8 의 내림 = 4
print(math.ceil(8.4)) # 8.4 의 올림 = 9
print(math.sqrt(4)) # 4 의 제곱근 = 2.0
```

▼ 2-4) 랜덤함수

- random 모듈: 난수를 생성해주는 모듈

```
from random import *

print(random()) # 0.0 이상 1.0 미만 범위 내 임의의 값 생성
```

```
print(random() * 10) # 0.0 이상 10.0 미만 범위 내 임의의 값 생성
print(int(random() * 10)) # 0 이상 10 미만 범위 내 임의의 정수 값 생성
print(int(random() * 10) + 1) # 1 이상 11 미만(10 이하) 범위 내 임의의 정수 값 생성
```

```
# 1 이상 46 미만(45 이하) 범위 내 임의의 정수 값 생성
print(int(random() * 45) + 1) # 1 이상 11 미만(10 이하) 범위 내 임의의 정수 값 생성
```

함수 이름	의미	예제
randrange	주어진 범위 내의 임의의 정수 값 생성	randrange(1, 46)
randint	주어진 범위 내의 임의의 정수 값 생성 (마지막 값 포함)	randint(1, 45)

```
print(randrange(1, 46)) # 1 이상 46 미만 범위 내 임의의 정수 값 생성
print(randint(1, 45)) # 1 이상 45 이하(45 포함) 범위 내 임의의 정수 값 생성
```

```
# How about like this?

print(randint(1, 45)) # 1 이상 45 이하(45 포함) 범위 내 임의의 정수 값 생성
print(randint(1, 45))
print(randint(1, 45))
print(randint(1, 45))
print(randint(1, 45))
print(randint(1, 45))
```

▼ 3. 문자열

문자열: 문자들의 집합

▼ 3-1) 문자열

```
sentence1 = '파이썬은 쉬워요'
print(sentence1)

sentence2 = "파이썬은 재미있어요"
print(sentence2)
```

```
sentence3 = """
파이썬은 쉽고,
재미있어요
"""
print(sentence3)
```

▼ 3-2) 슬라이싱

슬라이싱: 데이터 내 원하는 부분을 잘라서 가져오는 것

- 예시) 주민등록번호: 990804-1234567
  - 앞 6자리: 생년;월;일

생년		월		일	
9	9	0	8	0	4

  - 뒤 7자리: 성별; 지역번호; 검증번호

성별	지역번호					검증
1	2	3	4	5	6	7

- **변수명[인덱스]**
  - 인덱스에 해당하는 변수 값을 잘라서 가져오는 동작

    (인덱스 값은 일반적으로 0부터 시작)

```
id_num = "990804-1234567"

print("성별: " + id_num[7]) # 성별: 1
```

- **변수명[시작인덱스:종료인덱스]**
  - 시작인덱스부터 종료인덱스 전까지 값들을 잘라서 가져오는 동작

```
# id_num = "990804-1234567"

print("연: " + id_num[0:2]) # 0 부터 2 전까지 (0, 1): 99 년
print("월: " + id_num[2:4]) # 2 부터 4 전까지 (2, 3): 08 월
print("일: " + id_num[4:6]) # 4 부터 6 전까지 (4, 5): 04 일
```

그외 슬라이싱

- **변수명[:인덱스]**
  - 처음부터 인덱스 전까지 슬라이싱
- **변수명[인덱스:]**
  - 인덱스부터 끝까지 슬라이싱
- **변수명[:]**
  - 처음부터 끝까지 전체를 슬라이싱

```
# id_num = "990804-1234567"

print("생년월일: " + id_num[:6]) # 0 부터 6 전까지 = id_num[0:6]

print("뒤 7자리: " + id_num[7:]) # 7 부터 끝까지 = id_num[7:14]
print("뒤 7자리: " + id_num[-7:]) # 끝에서부터 앞으로 7번째까지
```

▼ 3-3) 문자열처리 함수

- 문자열처리에 유용한 함수 리스트

함수 이름	의미
lower	소문자로 변환
upper	대문자로 변환
isupper	대문자인지 확인
islower	소문자인지 확인
replace	문자열 바꾸기
index	찾으려는 문자열의 인덱스 (없으면 에러)
find	찾으려는 문자열의 인덱스 (없으면 -1)
count	문자열이 나온 횟수

- 소문자와 대문자가 섞인 문자열 처리에 관한 함수: lower(), upper(), isupper(), islower()

```
note = "Python is Easy and Fun"

print(note.lower()) # python is easy and fun
print(note.upper()) # PYTHON IS EASY AND FUN

print(note[0].isupper()) # True: 0 번째 인덱스의 값이 대문자인지 확인
print(note[1].islower()) # True: 1 번째 인덱스의 값이 소문자인지 확인

print(len(note)) # 22 (띄어쓰기를 포함한 문자열의 전체 길이)

print(note.replace("Python", "C++")) # C++ is Easy and Fun
```

- 문자열 내 특정 문자의 위치를 확인하는 함수: index(), find()

```
# note = "Python is Easy and Fun"

idx = note.index("a") # 처음으로 발견된 a 의 인덱스
print(idx) # 11 : Easy 의 a
idx = note.index("a", idx + 1) # 12 번째 인덱스 이후에 처음으로 발견된 a 의 인덱스
print(idx) # 15 : and 의 a

fidx = note.find("a") # 처음으로 발견된 a 의 인덱스
print(fidx) # 11 : Easy 의 a
fidx= note.find("a", fidx + 1) # 12 번째 인덱스 이후에 처음으로 발견된 a 의 인덱스
print(fidx) # 15 : and 의 a

print(note.index("C++")) # C++ 문자가 없기 때문에 에러가 발생하고, 프로그램 종료
print(note.find("C++")) # C++ 문자가 없으면 -1 을 반환하고, 프로그램 계속 수행
```

- 문자열 내 특정 문자의 사용 빈도 수: count()

```
# note = "Python is Easy and Fun"

print(note.count("a")) # 2: 문자열 내에서 a 가 나온 횟수
```

▼ 3-4) 문자열포멧

다양한 방식의 문자열 출력 방법들 존재

- **%**

  print("문자열 **%d** 문자열" **% 정수**)

  print("문자열 **%c** 문자열" **% 문자**)

  print("문자열 **%s** 문자열" **% 문자열**)

```
print("올해는 %d년 입니다." % 2022) # 올해는 2022년 입니다.
print("올해는 %s의 해 입니다." % "검은 호랑이") # 올해는 검은 호랑이의 해 입니다.
print("Tiger는 %c로 시작합니다." % "T") # Tiger는 T로 시작해요.

print("올해는 %s년 입니다." % 2022) # 올해는 2022년 입니다.

print("내년은 %d년, %s의 해 입니다." %(2023, "검은 토끼")) # 내년은 2023년, 검은 토끼의 해 입니다.
```

- **.format()**

```
print("올해는 {}년 입니다.".format(2022)) # 올해는 2022년 입니다.
print("올해는 {}의 해 입니다.".format("검은 호랑이")) # 올해는 검은 호랑이의 해 입니다.
print("내년은 {}년, {}의 해 입니다.".format(2023, "검은 토끼")) # 내년은 2023년, 검은 토끼의 해 입니다.
```

- **{이름}**

```
print("올해는 {year}년 입니다.".format(year=2022)) # 올해는 2022년 입니다.
print("올해는 {color} {zodiac}의 해 입니다.".format(color="검은",zodiac="호랑이")) # 올해는 검은 호랑이의 해 입니다.
print("내년은 {year}년, {color} {zodiac}의 해 입니다.".format(year=2023, color="검은",zodiac="토끼")) # 내년은 2023년, 검은 토끼의 해 입니다.
```

- **f-string (Python 3.6 이상)**

```
year = 2023
color = "검은"
zodiac = "토끼"

print(f"내년은 {year}년, {color} {zodiac}의 해 입니다.") # 내년은 2023년, 검은 토끼의 해 입니다.
```

▼ 3-5) 탈출문자(\)

탈출문자	의미
\n	줄바꿈
\t	8칸 이동(Tab)
\"	큰 따옴표 출력
\'	작은 따옴표 출력
\\	\ 출력
r	문자열 그대로 출력(탈출문자 무시)

```
# 군 장병 AI/SW 역량강화사업 Python/R 기초
print("군 장병 AI/SW 역량강화사업 Python/R 기초")

# 군 장병 AI/SW 역량강화사업
# Python/R 기초
print("군 장병 AI/SW 역량강화사업\nPython/R 기초")

# 군 장병 AI/SW 역량강화사업        Python/R 기초
print("군 장병 AI/SW 역량강화사업\tPython/R 기초")

# 저는 강의를 맡은 "김학구" 입니다.
# print("저는 강의를 맡은 "김학구" 입니다.") # Error
print('저는 강의를 맡은 "김학구" 입니다.')
print("저는 강의를 맡은 \"김학구\" 입니다.")

# 저는 강의를 맡은 '김학구' 입니다.
print("저는 강의를 맡은 \'김학구\' 입니다.")


# C:\Users\Python\LectureNotes
# print("C:\Users\Python\LectureNotes") # Error
print("C:\\Users\\Python\\LectureNotes")
print(r"C:\Users\Python\LectureNotes")
```

▼ 4. 자료형(심화)

▼ 4-1) 리스트(list)

숫자와 문자열만으로 프로그래밍을 하기엔 부족한 점이 많다.

예시) 1부터 10까지의 숫자 중 홀수 모음인 1, 3, 5, 7, 9의 집합을 생각해 보면, 이런 숫자 모음을 숫자나 문자열로 표현하기는 쉽지 않다.

리스트는 이러한 불편함을 해소할 수 있는 자료형: **[value1, value2, …]**

```
# 지하철 칸마다 탑승객의 수: 10명, 20명, 30명
# Without list
subway1 = 10
subway2 = 20
subway3 = 30

# With list
subway = [10, 20, 30]
print(subway) # [10, 20, 30]
```

```
# 지하철 칸별 탑승객: 백준기, 최종원, 김학구
subway = ["백준기", "최종원", "김학구"]
print(subway) # ['백준기', '최종원', '김학구']

# 김학구 교수가 타고있는 지하철 칸: index() 함수 활용
print(subway.index("김학구")) # 2
```

- append(): 리스트의 마지막에 데이터를 추가

```
# 다음 역에서 김영빈 교수님이 지하철 다음 칸에 탐
subway.append("김영빈")
print(subway) # ['백준기', '최종원', '김학구', '김영빈']
```

- insert(): 인덱스 값을 이용해서 원하는 위치에 데이터를 삽입

```
# 조진혁 박사님이 백준기 교수님과 최종원 교수님 사이에 탐
subway.insert(1, "조진혁")
print(subway) # ['백준기', '조진혁', '최종원', '김학구', '김영빈']
```

- pop(): 리스트 내 맨 뒤에 있는 데이터를 하나씩 제거

```
# 김영빈 교수님 지하철 하차
print(subway.pop())
print(subway) # ['백준기', '조진혁', '최종원', '김학구']

# 김학구 교수 지하철 하차
print(subway.pop())
print(subway) # ['백준기', '조진혁', '최종원']

# 최종원 교수님 지하철 하차
print(subway.pop())
print(subway) # ['백준기', '조진혁']
```

다양한 리스트 관리: sort(), reverse(), clear()

```
list_num = [4, 8, 1, 3, 5]

# 오름차수 정렬
list_num.sort()
print(list_num) # [1, 3, 4, 5, 8]

# 현재 순서를 반대로 정렬
list_num.reverse()
print(list_num) # [8, 5, 4, 3, 1]

# 리스트 내 데이터 모두 지우기
list_num.clear()
print(list_num) # []
```

리스트 활용: 다양한 자료형들을 리스트에서 함께 관리

```
# 다양한 자료형들을 리스트에서 관리
list_var = ["김학구", 2021, True]
print(list_var) # ['김학구, 2021, True']

# 리스트의 확장
list_num = [4, 8, 1, 3, 5] # list_num 재정의
list_num.extend(list_var) # list 확장
print(list_num) # [4, 8, 1, 3, 5, "김학구", 2021, True]
```

▼ 4-2) 튜플(tuple)

튜플: 리스트와 유사하지만 편집 불가 (리스트의 '읽기 전용' 버전). 처음 정의할 때를 빼고는 데이터 변경이나 추가, 삭제 등 편집이 불가능. 반면에 연산 속도가 빠름: **(value1, value2, …)**

```
cau_ai = ("백준기", "김영빈", "최종원")
print(cau_ai[0]) # 백준기
print(cau_ai[1]) # 김영빈
print(cau_ai[2]) # 최종원
```

```
(name, department, year) = ("김학구", "첨단영상대학원", 2021)
print(name, department, year) # 김학구 첨단영상대학원 2021
```

▼ 4-3) 사전(dictionary)

사전(dictionary)은 key(단어)와 value(뜻)의 쌍으로 구성: **{key1 : value1, key2 : value2, …}**

```
# 사전을 이용한 연구실 예제
office = {801: "최종원", 818: "김학구"}

# 각 연구실별 교수 확인
print(office[801]) # 최종원
print(office[818]) # 김학구

# get() 함수를 이용하여 확인
print(office.get(801)) # 최종원
print(office.get(818)) # 김학구

# []와 get() 함수 사용의 차이
# print(office[810]) # Error
print(office.get(810)) # None

# 연구실별 교수 배정 또는 빈 공간 확인
print(818 in office) # True
print(810 in office) # False
```

```
# key: 문자열도 가능
office = {"305관 801호": "최종원", "305관 818호": "김학구"}

# 각 연구실별 교수 확인
print(office["305관 801호"]) # 최종원
print(office["305관 818호"]) # 김학구

# 연구실 교수 배정 변경
print(office) # {'305관 801호': '최종원', '305관 818호': '김학구'}
office["305관 818호"] = "김영빈"
office["305관 810호"] = "김학구"
print(office) # {'305관 801호': '최종원', '305관 818호': '김영빈', '305관 810호': '김학구'}

# 교수 은퇴, 연구실 반납
del office["305관 810호"]
print(office) # {'305관 801호': '최종원', '305관 818호': '김영빈'}
```

```
# 현재 이용 중인 건물 내 연구실 확인: keys()
print(office.keys()) # dict_keys('305관 801호', '305관 818호')

# 현재 활동 중인 교수님 확인: values()
print(office.values()) # dict_values('최종원', '김영빈')

# 어떤 연구실을 어떤 교수님이 쓰는 지 확인: items()
print(office.items()) # dict_items('305관 801호', '305관 818호') # dict_items([('305관 801호', '최종원'), ('305관 818호', '김영빈')])

# 전체 삭제
office.clear()
print(office) # {}
```

▼ 4-4) 세트(set)

세트(집합): 중복을 허용하지 않으며, 데이터의 순서도 보장하지 않음: **{value1, value2, …}**

```
# 세트: 중복을 허용하지 않음
temp_set = {1, 2, 3, 3, 3}
print(temp_set) # {1, 2, 3}

# {} 대신 set() 사용 가능
cau_ai = {"백준기", "이재성", "김영빈", "최종원"} # AI대학원 소속
cau_gsaim = set(["백준기", "김영빈", "최종원", "김학구"]) # 첨단영상대학원 소속
```

```
# 교집합: &, intersection()
print(cau_ai & cau_gsaim) # {'백준기', '김영빈', '최종원'}
print(cau_ai.intersection(cau_gsaim)) # {'백준기', '김영빈', '최종원'}

# 합집합: |, union()
print(cau_ai | cau_gsaim) # {'백준기', '이재성', '김영빈', '최종원', '김학구'}
print(cau_ai.union(cau_gsaim)) # {'백준기', '이재성', '김영빈', '최종원', '김학구'}

# 차집합: -, difference()
print(cau_ai - cau_gsaim) # {'이재성'}
print(cau_ai.difference(cau_gsaim)) # {'이재성'}
```

```
# AI 대학원 새로운 교원 추가
cau_ai.add("김인공")
print(cau_ai) # {'이재성', '김인공'}

# 첨단영상대학원 김학구 교수 은퇴
cau_gsaim.remove("김학구")
print(cau_gsaim) # {'백준기', '김영빈', '최종원'}
```

▼ 4-5) 자료구조의 변경

자료구조의 변경: 리스트→튜플, 튜플→세트, 세트→리스트 등 자유롭게 변환 가능

```
# 자료형 확인: type()
menu = {"돈까스", "치킨", "짜장면"} # 세트 형태
print(menu, type(menu)) # {'돈까스', '치킨', '짜장면'} <class 'set'>

# 세트 -> 리스트
menu = list(menu) # 리스트 형태로 변환
print(menu, type(menu)) # {'돈까스', '치킨', '짜장면'} <class 'list'>

# 리스트 -> 튜플
menu = tuple(menu) # 튜플 형태로 변환
print(menu, type(menu)) # {'돈까스', '치킨', '짜장면'} <class 'tuple'>

# 튜플 -> 세트
menu = set(menu) # 세트 형태로 변환
print(menu, type(menu)) # {'돈까스', '치킨', '짜장면'} <class 'set'>
```