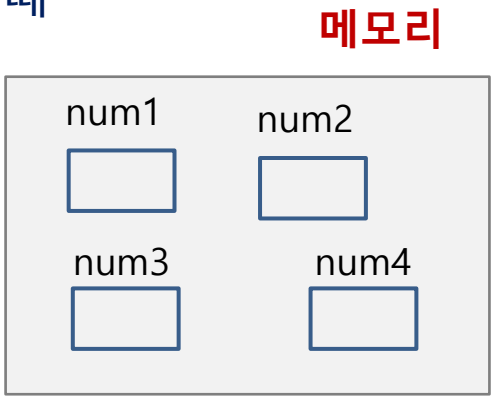
3장. 자료구조(리스트, 딕셔너리, 튜플)

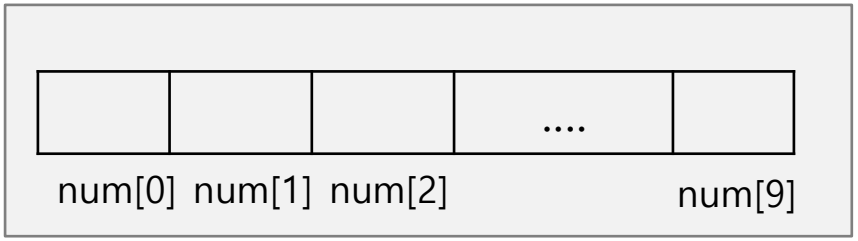
리스트(배열) 사용의 필요성

- **리스트(배열) 사용의 필요성**
  - 정수 10개를 이용한 학생의 성적 프로그램을 만들때 10개의 변수를 선언

    (num1, num2, num3… num10)
  - 정보가 흩어진 채 저장되고, 변수 이름이 많아 비효율적이고 관리하기 어렵다.



- **리스트(배열) 사용의 장점**
  - 인덱스를 이용하여 순차(순서)적으로 관리할 수 있다 -> 효율적이다.



리스트(list)란?

- **리스트(list)란?**

  - 여러 개의 연속적인 값을 저장하고자 할 때 사용하는 자료형이다.
    (변수는 1개의 값만을 저장하고 변경할 수 있다.)
  - 리스트의 맨 앞요소의 위치(인덱스)는 0번이다.

- **리스트의 생성**

  리스트 이름 = [요소1, 요소2, 요소3...]

  season = [“봄”, “여름”, “가을”, “겨울”]

  number = [1. 2, 3, 4, 5]

리스트(list)의 활용

- 정수형 리스트

```
# 리스트 생성
my_list = [1, 2, 3, 4, 5]

# 리스트 출력
print("리스트:", my_list)
print(type(my_list))

# 리스트의 길이 출력
print("리스트의 길이:", len(my_list))

# 리스트의 특정 요소 접근
print("리스트의 첫 번째 요소:", my_list[0])
print("리스트의 세 번째 요소:", my_list[2])
print("리스트의 마지막 요소:", my_list[-1])

# 리스트의 슬라이싱(':' 연산자 사용-부분 리스트 추출)
print("리스트의 첫 세 요소:", my_list[0:3])
print("리스트의 마지막 두 요소:", my_list[-2:])
print("리스트의 모든 요소:", my_list[:])
```

리스트(list)의 활용

- 정수형 리스트

```
# 리스트 요소 변경
my_list[1] = 20
print("요소 변경 후 리스트:", my_list)

# 리스트에서 요소 제거 - remove() 메서드 사용
del my_list[2]
print("요소 제거 후 리스트:", my_list)

# 리스트에 요소 추가 - append() 메서드 사용
# my_list.append(6)
# print("요소 추가 후 리스트:", my_list)
```

```
리스트: [1, 2, 3, 4, 5]
<class 'list'>
리스트의 길이: 5
리스트의 첫 번째 요소: 1
리스트의 세 번째 요소: 3
리스트의 마지막 요소: 5
리스트의 첫 세 요소: [1, 2, 3]
리스트의 마지막 두 요소: [4, 5]
리스트의 모든 요소: [1, 2, 3, 4, 5]
요소 변경 후 리스트: [1, 20, 3, 4, 5]
요소 제거 후 리스트: [1, 20, 4, 5]
```

리스트(list)의 생성

- 문자형 리스트

```
# 문자형 리스트
carts = ['라면', '계란', '커피', '딸기']

print("리스트:", carts)
print("리스트의 길이:", len(carts))

# 리스트의 특정 요소 접근
print("리스트의 첫 번째 요소:", carts[0])
print("리스트의 세 번째 요소:", carts[2])
print("리스트의 마지막 요소:", carts[-1])

# 리스트 요소 변경
carts[1] = '우유'
print("요소 변경 후 리스트:", carts)
```

```
리스트: ['라면', '계란', '커피', '딸기']
리스트의 길이: 4
리스트의 첫 번째 요소: 라면
리스트의 세 번째 요소: 커피
리스트의 마지막 요소: 딸기
요소 변경 후 리스트: ['라면', '우유', '커피', '딸기']
```

리스트(list) 반복 – in 사용

- **for 변수 in 리스트:**

```
my_list = [10, 20, 30, 40, 50]
print(20 in my_list)  # True
print(25 in my_list)  # False

print("for문을 사용한 리스트 출력:")

for item in my_list:
    print(item, end=' ')
print()

# 40보다 큰 요소만 출력
print("40보다 큰 요소 출력:")

for item in my_list:
    if item > 40:
        print(item, end=' ')
print()
```

리스트(list)의 연산

◆ 리스트의 연산 – 합계, 평균, 최대값, 최소값 구하기

```
# 리스트의 요소 합계 계산
total = 0
for item in my_list:
    total += item
print("리스트 요소의 합계:", total)

# 리스트 요소의 평균
average = total / len(my_list)
print("리스트 요소의 평균:", average)

# 리스트 요소 중 최대값과 최소값 찾기
max_value = my_list[0]
min_value = my_list[0]
for item in my_list:
    if item > max_value:
        max_value = item
    if item < min_value:
        min_value = item
print("리스트 요소 중 최대값:", max_value)
print("리스트 요소 중 최소값:", min_value)
```

리스트(list) 반복 – in 사용

- **if 변수 in [list]**

  리스트 내부에 값이 있으면 True, 없으면 False

```
# 음식 분류하기 - 한식, 일식, 중식
foods = ["비빔밥", "짜장면", "초밥", "김치찌게"]

for food in foods:
    if food in ["짜장면",  "짬뽕"]:
        print(f'{food}는(은) 중식입니다.')
    elif food in ["초밥", "우동"]:
        print(f'{food}는(은) 일식입니다.')
    else:
        print(f'{food}는(은) 한식입니다.')
```

```
비빔밥는(은) 한식입니다.
짜장면는(은) 중식입니다.
초밥는(은) 일식입니다.
김치찌게는(은) 한식입니다.
```

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

함수	기능	사용 예
append()	요소 추가	a = [1, 2, 3] a.append(4) a = [1, 2, 3, 4]
insert()	특정 위치에 추가	a = [2, 4, 5] a.insert(1,3) #1번 위치에 3 삽입 a = [2, 3, 4, 5]

pop()	요소 삭제	a = [1, 2, 3, 4, 5]
		a.pop() # 마지막 위치의 요소 제거
		a = [1, 2, 3, 4]
		a.pop(1) #1 위치의 2 제거
		a = [1, 3, 4]

remove()	특정 요소 삭제	s = ['모닝', 'BMW', 'BENZ', '스포티지']
		s.remove('BMW') #요소 직접 삭제
		s = ['모닝', 'BENZ', '스포티지']

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

함수	기능	사용 예
sort()	정렬	a = [1, 4, 2, 3] a.sort() [1, 2, 3, 4]
reverse()	뒤집기	lower = ['b', 'c', 'a'] lower.reverse() ['a', 'b', 'c']
extend(리스트)	리스트의 끝에 리스트 추가	li = ['a', 'b'], li.extend(['c','d']) ['a', 'b', 'c', 'd'],
copy()	리스트 복사	n = [1, 2, 3] m = n.copy()

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

```
# 요소 추가
fruits = ['사과', '바나나', '귤']
fruits.append('포도')  # 맨 끝에 요소 추가
print("append 후:", fruits)

# 요소 삽입
fruits.insert(1, '딸기')  # 인덱스 1에 요소 삽입
print("insert 후:", fruits)

# 요소 제거
fruits.remove('바나나')  # 값으로 요소 제거
print("remove 후:", fruits)

# 맨 마지막 요소 제거
last_fruit = fruits.pop()  # 맨 마지막 요소 제거 및 반환
print("pop 후:", fruits)
```

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

```
# 요소 정렬
numbers = [5, 2, 9, 1, 5, 6]
numbers.sort()  # 오름차순 정렬
print("sort 후:", numbers)

# 요소 뒤집기
numbers.reverse()  # 리스트 뒤집기
print("reverse 후:", numbers)

# 리스트 복사
copied_numbers = numbers.copy()  # 리스트 복사
print("copy 후:", copied_numbers)

# 리스트 확장
numbers.extend([10, 11, 12])  # 리스트 확장
print("extend 후:", numbers)
```

리스트(list) 복사

◆ 리스트의 복사 – append() 사용

```
# append() 사용하여 리스트 복사
a1 = [1, 2, 3, 4, 5]
a2 = []

for item in a1:
    a2.append(item)
print("a2 =", a2)

a3 = []
# a1 요소 중 홀수만 a3에 추가
for item in a1:
    if item % 2 == 1:
        a3.append(item)
print("a3 =", a3)
```

리스트(list) 내포

◆ 리스트 내포 사용하기

[표현식 for 항목(요소) in 리스트]

```
a4 = [item for item in a1]
print("a4 =", a4)

a5 = [item for item in a1 if item % 2 == 1]
print("a5 =", a5)
```

문자열 인덱싱, 슬라이싱

◆ 문자열은 특별한 1차원 리스트이다.

문자열(시작번호:끝번호)

※ 끝번호는 (끝번호 -1)과 같다

```
s = "Hello, World!"
print("문자열:", s)
print("문자열의 길이:", len(s))

# 문자열의 특정 문자 접근
print("첫 번째 문자:", s[0])
print("일곱 번째 문자:", s[6])
print("마지막 문자:", s[-1])

# 문자열 슬라이싱
print("처음 다섯 문자:", s[0:5])
print("쉼표부터 끝까지:", s[5:])
print("모든 문자:", s[:])
```

```
문자열: Hello, World!
문자열의 길이: 13
첫 번째 문자: H
일곱 번째 문자:
마지막 문자: !
처음 다섯 문자: Hello
쉼표부터 끝까지: , World!
모든 문자: Hello, World!
```

문자열 함수

- **문자열 함수(메서드) 정리**

메서드	설명
split()	s = 'banana, grape, kiwi' s = fruit.split(',') [구분기호로 나누고 리스트로 만듬] s ['banana', ' grape', ' kiwi']
replace()	s = 'Hello, World' s = s.replace('World', 'Korea') [문자를 변경함] 'Hello, Korea'
find()	s = "Hello" s.find('H') 0 s.find('k') -1 [문자열이 존재하는 위치 반환. 없으면 -1반환]
strip()	s = " Hi, lee" s.strip() Hi, lee

문자열 함수

- 문자열 함수(메서드)

```
# 대소문자 변환
print("소문자 변환:", s.lower()) # hello, world!
print("대문자 변환:", s.upper()) # HELLO, WORLD!

# 문자 개수 세기
print("문자열에서 'o'의 개수:", s.count('o')) # 2

# 특정 문자 위치 찾기
print("문자열에서 'World'의 위치:", s.find('World')) # 7

# 문자열 교체
print("문자열 교체:", s.replace('World', 'Python')) # Hello, Python!
```

문자열 함수

- 문자열 함수(메서드)

```
# split과 join
csv = "apple,banana,cherry"
fruits = csv.split(',')
print("분리된 과일 리스트:", fruits) # ['apple', 'banana', 'cherry']
# apple,banana,cherry
print("과일 리스트를 다시 문자열로:", ",".join(fruits))

# strip
msg = "   Good Luck!   "
print("양쪽 공백 제거:", msg.strip()) # "Good Luck!"
print("왼쪽 공백 제거:", msg.lstrip()) # "Good Luck!   "
print("오른쪽 공백 제거:", msg.rstrip()) # "   Good Luck!"
```

실습 문제 – 문자열 처리

2025년 민생회복 지원금 신청 프로그램을 아래와 같이 구현하세요.

출생연도 끝자리	접종 요일
1 또는 6	월요일
2 또는 7	화요일
3 또는 8	수요일
4 또는 9	목요일
5 또는 0	금요일

☞ 실행 결과

```
출생연도 입력: 2023
신청일은 수요일입니다.
```

```
출생연도 입력: 2010
출생연도는 1900년부터 2006년 사이여야 합니다.
```

2차원 리스트(list)

- **2차원 리스트의 선언 및 생성**
  - 리스트 내부에 리스트를 가진 자료 구조이다.
  - 행과 열의 표(테이블) 형태를 이루고 있다.

리스트 이름 = [요소1, 요소2, **[요소1, 요소2, 요소3]**]

	열1	열2
행1	a[0][0]	a[0][1]
행2	a[1][0]	a[1][1]
행3	a[2][0]	a[2][1]

2차원 리스트(list)

- 2차원 리스트 생성 및 출력

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# 2차원 리스트의 크기
print("2차원 리스트의 크기:")
print("행의 수:", len(matrix))  # 3
print("첫 번째 행의 열의 수:", len(matrix[0]))  # 3
print("두 번째 행의 열의 수:", len(matrix[1]))  # 3

print("2차원 리스트의 요소 접근:")
print("첫 번째 행, 두 번째 열:", matrix[0][1])  # 2
print("세 번째 행, 첫 번째 열:", matrix[2][0])  # 7
print("두 번째 행 전체:", matrix[1]) # [4, 5, 6]

print("2차원 리스트 (행렬):")
for row in matrix:
    print(row)
```

2차원 리스트(list)

```
# 요소 추가
matrix.append([10, 11, 12])  # 새로운 행 추가
print("새로운 행 추가 후:")
for row in matrix:
    print(row)

# 요소 수정
matrix[0][0] = 99  # 첫 번째 행, 첫 번째 열 수정
print("첫 번째 행, 첫 번째 열 수정 후:")
for row in matrix:
    print(row)

# 요소 삭제
del matrix[1]  # 두 번째 행 삭제
print("두 번째 행 삭제 후:")
for row in matrix:
    print(row)
```

2차원 리스트(list)

- 2차원 리스트의 연산

```
d = [
    [10, 20],
    [30, 40],
    [50, 60, 70]
]

total = 0  # 총합 변수
count = 0  # 요소 개수 변수

for row in d:
    for value in row:
        total += value
        count += 1

# 평균 계산
average = total / count

print("총합:", total)  # 총합: 280
print("요소 개수:", count) # 요소 개수: 7
print("평균:", average) # 평균: 40.0
```

딕셔너리(Dictionary)

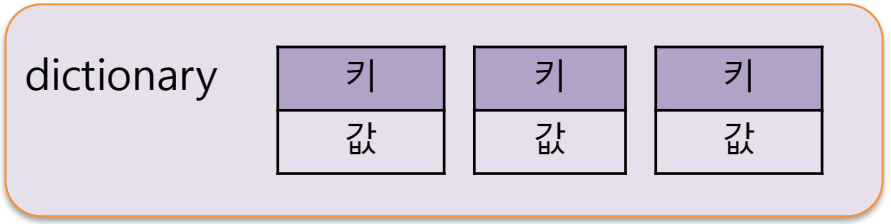
◆ 딕셔너리

리스트 처럼 여러 개의 값을 저장할 수 있고, 키(key)와 값(value)으로 대응시켜 저장하는 자료구조이다.

중괄호{ }를 사용한다.

딕셔너리 이름 = **{ 키:값, 키:값....}**

{ 'name': '한국민', 'age': 28 }



딕셔너리(Dictionary)

◆ 딕셔너리 주요 메서드

함수	사용 예
d[key] = value	d = {'Tomas':13, 'Jane':9} d['Mike'] = 10 **# 요소 추가** {'Tomas':13, 'Jane':9, 'Mike':10 }
del d[key]	del d['Jane'] **#요소 삭제** {'Tomas':13, 'Mike':10 }
d.pop(key)	d.pop('Mike') 10 {'Tomas':13}
clear()	d.clear() # d={ } 빈 딕셔너리
d.keys()	d.keys() **# 모든 키 가져오기** d_keys(['Tomas', 'Mike'])
d.values()	d.Values() **# 모든 값 가져오기** d_values([13, 10])
d.get(key)	d.get('Jane') #9

딕셔너리(Dictionary)

◆ 딕셔너리 생성 및 관리

```
student = {
    "name": "한강",
    "age": 21,
    "major": "컴퓨터 공학"
}

print("딕셔너리 내용:", student)
print(type(student))  # <class 'dict'>

print("딕셔너리의 크기:", len(student))  # 3

print("딕셔너리의 요소 접근:")
print("이름:", student["name"])  # 한강
print("나이:", student["age"])    # 21
print("전공:", student["major"])  # 컴퓨터 공학

# 딕셔너리 요소 추가
student["university"] = "한강대학교"
print("요소 추가 후 딕셔너리:", student)
```

딕셔너리(Dictionary)

```python
# 딕셔너리 요소 수정
student["age"] = 22
print("요소 수정 후 딕셔너리:", student)

# 딕셔너리 요소 삭제
# del student["major"]
student.pop("major")  # pop 메서드를 사용하여 요소 삭제
print("요소 삭제 후 딕셔너리:", student)


# 키 목록과 값 목록
keys = student.keys()
values = student.values()
print("키 목록:", list(keys))
print("값 목록:", list(values))

print("딕셔너리 순회:")
for key, value in student.items():
    print(f"{key}: {value}")
```

딕셔너리(Dictionary)

```
딕셔너리 내용: {'name': '한강', 'age': 21, 'major': '컴퓨터 공학'}
<class 'dict'>
딕셔너리의 크기: 3
딕셔너리의 요소 접근:
이름: 한강
나이: 21
전공: 컴퓨터 공학
요소 추가 후 딕셔너리: {'name': '한강', 'age': 21, 'major': '컴퓨터 공학', 'university': '한강대학교'}
요소 수정 후 딕셔너리: {'name': '한강', 'age': 22, 'major': '컴퓨터 공학', 'university': '한강대학교'}
요소 삭제 후 딕셔너리: {'name': '한강', 'age': 22, 'university': '한강대학교'}
키 목록: ['name', 'age', 'university']
값 목록: ['한강', 22, '한강대학교']
딕셔너리 순회:
name: 한강
age: 22
university: 한강대학교
```

딕셔너리(Dictionary)

- 용어 사전 만들기

```
♠ 컴퓨터 용어 사전 ♠

검색할 용어를 입력하세요(종료: q or Q): 이진수
컴퓨터가 사용하는 0과 1로 이루어진 수
검색할 용어를 입력하세요(종료: q or Q): 버그
프로그램이 적절하게 동작하는데 실패하거나 오류가 발생하는 코드 조각
검색할 용어를 입력하세요(종료: q or Q): 함수
정의된 단어가 없습니다.
검색할 용어를 입력하세요(종료: q or Q): q
프로그램 종료!
```

1. Dictionary 자료구조에 컴퓨터 용어와 정의를 저장한다.
2. 용어를 계속 반복해서 검색 할 수 있다.
3. 검색한 용어가 없으면 정의된 단어가 없음을 알려준다.
4. 검색을 종료하려면 'q' 또는 'Q'를 입력한다.

딕셔너리(Dictionary)

- 용어 사전 만들기

```
print("♠ 컴퓨터 용어 사전 ♠")

dic = {
    "CPU": "Central Processing Unit - 컴퓨터의 중앙 처리 장치",
    "RAM": "Random Access Memory - 임의 접근 메모리",
    "이진수": "0과 1로 이루어진 수 체계",
    "알고리즘": "문제를 해결하기 위한 절차나 방법"
}
```

딕셔너리(Dictionary)

- 용어 사전 만들기

```
while True:
    word = input("검색할 용어를 입력하세요 (종료: q or Q): ")
    if word == 'q' or word == 'Q':
        print("용어 사전 프로그램을 종료합니다.")
        break
    definition = dic.get(word) # 용어에 대한 정의 검색
    if definition:
        print(f"{word}: {definition}")
    else:
        print(f"'{word}' 용어는 사전에 없습니다.")
```

학생의 성적 관리

- 학생의 성적 통계

```
student_list = [
    {"name": "이대한", "kor": 85, "eng": 78, "math": 92},
    {"name": "박민국", "kor": 92, "eng": 88, "math": 79},
    {"name": "오상식", "kor": 78, "eng": 85, "math": 80},
    {"name": "최지능", "kor": 90, "eng": 92, "math": 88},
]

print("첫번째 요소 검색", student_list[0])
print("첫번째 학생 이름:", student_list[0]['name'])  # 이대한

print("\n전체 학생 성적:")
print("이름\t국어\t영어\t수학")
for student in student_list:
    print(f"{student['name']}\t{student['kor']}\t \
{student['eng']}\t{student['math']}")
```

학생의 성적 관리

- 학생의 성적 통계

```
# 성적 통계 계산
total_kor = 0 # 국어 총점
total_eng = 0 # 영어 총점
total_math = 0 # 수학 총점

# 각 과목별 총점 계산
for student in student_list:
    total_kor += student['kor']
    total_eng += student['eng']
    total_math += student['math']

num_students = len(student_list)  # 학생 수
avg_kor = total_kor / num_students # 국어 평균
avg_eng = total_eng / num_students # 영어 평균
avg_math = total_math / num_students # 수학 평균

print("\n성적 평균:")
print(f"국어: {avg_kor:.1f}, 영어: {avg_eng:.1f}, 수학: {avg_math:.1f}")
```

```
첫번째 요소 검색 {'name': '이대한', 'kor': 85, 'eng': 78, 'math': 92}
첫번째 학생 이름: 이대한

전체 학생 성적:
이름    국어    영어    수학
이대한  85      78      92
이대한  85      78      92
박민국  92      88      79
박민국  92      88      79
오상식  78      85      80
오상식  78      85      80
최지능  90      92      88
최지능  90      92      88

성적 평균:
국어: 86.2, 영어: 85.8, 수학: 84.8
```

실습 문제 – 딕셔너리

다음의 실행 결과가 나오도록 빈 칸을 작성하시오.(파일: member.py)

```
member = {"이름": "신유빈", "나이": 20, "특기": "탁구"}
result = 

print(member)
print(result)
```

☞ 실행 결과

```
{'이름': '신유빈', '특기': '탁구'}
20
```

튜플(tuple)

- **튜플(tuple)**
  - 튜플의 요소를 변경(추가, 수정, 삭제)할 수 없다.
  - 요소 추가는 초기화나 튜플간 합치기를 하면 가능함
  - 리스트처럼 동일한 방식으로 인덱싱과 슬라이싱 가능함
  - 소괄호( ) 를 사용한다.

튜플 이름 = **(요소1, 요소2....)**

```
t1 = ()
t2 = (1, )
t3 = (1, 2, 3)
t4 = ('a', 'b', 'c')
```

튜플(tuple)

- 튜플 자료형

```python
# 튜플 생성
t = (1, 2, 3)
print(t)  # (1, 2, 3)
print(type(t))  # <class 'tuple'>

# 특정 요소 접근
print("첫 번째 요소:", t[0])  # 1
print("두 번째 요소:", t[1])  # 2
print("마지막 요소:", t[-1])  # 3

# 튜플 길이
print("튜플의 길이:", len(t))  # 3

# 슬라이싱
print("첫 두 요소:", t[0:2])  # (1, 2)
print("두 번째 요소부터 끝까지:", t[1:])  # (2, 3)
```

```
(1, 2, 3)
<class 'tuple'>
첫 번째 요소: 1
두 번째 요소: 2
<class 'tuple'>
첫 번째 요소: 1
두 번째 요소: 2
첫 번째 요소: 1
두 번째 요소: 2
두 번째 요소: 2
마지막 요소: 3
마지막 요소: 3
튜플의 길이: 3
첫 두 요소: (1, 2)
두 번째 요소부터 끝까지: (2, 3)
단일 요소 튜플: (5,)
튜플 연결: (1, 2, 3, 5)
```

튜플(tuple)

- 튜플 자료형

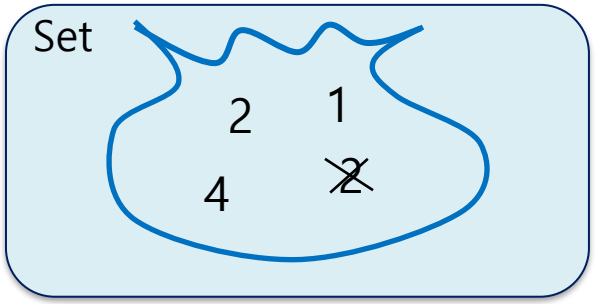
```
# 튜플은 변경 불가능(immutable)
# t[0] = 10  # 오류 발생

# 튜플 삭제 불가능
# del t[1]  # 오류 발생

#  단일 요소 튜플 생성
t2 = (5)
print("단일 요소 튜플 아님:", t2)  # 5

t2 = (5,)
print("단일 요소 튜플:", t2)  # (5,)

# 튜플 연결
t3 = t + t2
print("튜플 연결:", t3)  # (1, 2, 3, 5)
```

튜플의 요소는 수정 및 삭제 할 수 없다.

집합 자료형(set)

- 집합 자료형
  - 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료구조
  - 중복을 허용하지 않고 , 순서가 없다.
  - 중괄호 { }를 사용한다.

집합 이름 = { 요소1, 요소2, 요소3 }



집합(set) 관련 메서드

- set 관련 메서드

함수	사용 예
add(x)	s = {1, 2, 3} s.add(4) # 요소 추가 {1, 2, 3, 4}
remove(x)	s1 = {1, 3, 4} s1.remove(3) {1, 2}

clear()	s1 = {1, 3, 4}
	s1.clear() # 모두 지우기
	set()

x in s	fruits = {"apple", "banana", "grape"} "apple" in fruits True "grape" not in fruits #False

집합 자료형(set)

- 집합 자료형 예제

```
# 선언 방법
s = {1, 2, 3}
print(s)  # {1, 2, 3}
print(type(s))  # <class 'set'>

# 요소 추가
s.add(4)
print("요소 추가 후:", s)  # {1, 2, 3, 4}

# 중복 요소 추가 시도
s.add(3)
print("중복 요소 추가 시도 후:", s)  # {1, 3, 4} (변화 없음)

# 요소 제거
s.remove(2)
print("요소 제거 후:", s)  # {1, 3, 4}
```

집합 자료형(set)

- 집합 연산자

연산자	집합
&	교집합
|	합집합
a-b	차집합

```
# 집합 연산자
a = {1, 2, 3}
b = {2, 3, 4}

c = a & b  #교집합
d = a | b  #합집합
e = b - a  #차집합

print(c) #{2, 3}
print(d) #{1, 2, 3, 4}
print(e) #{4}
```

집합 자료형(set)

- 집합 자료형 예제

```
# 집합 연산
s2 = {3, 4, 5, 6}
print("두 집합:", s, s2)

# 합집합
union_set = s.union(s2)
print("합집합:", union_set)  # {1, 3, 4, 5, 6}

# 교집합
intersection_set = s.intersection(s2)
print("교집합:", intersection_set)  # {3, 4}
```

집합 자료형(set)

- 집합 자료형 예제

```
# 빈 집합 생성
s3 = set()
print(s3)  # set()

# 요소 추가
s3.add(10)
s3.add(20)
s3.add(10)  # 중복 추가 시도
print(s3)  # {10, 20}

# 리스트에서 집합으로 변환 (중복 제거)
lst = [1, 2, 2, 3, 4, 4]
unique_set = set(lst)
print(unique_set)  # {1, 2, 3, 4}
```