# 2024-1
# 기초 스터디

5. 튜플, 딕셔너리, 셋

# 목차

# 튜플

- 순서가 있는 데이터의 열거

  → **수정할 수 없는 리스트**

# 튜플 – 생성

- tuple(데이터그룹)

```
>>> a = tuple()
>>> a
()
```

```
>>> c = [1, 2, 3]
    a = tuple(c)
>>> a
(1, 2, 3)
```

- (x1, x2, x3, ...)

```
>>> a = (1, 2, 3)
>>> type(a)
<class 'tuple'>
```

```
>>> a = (1,)
>>> type(a)
<class 'tuple'>
```

```
>>> a = (1)
>>> type(a)
<class 'int'>
```

# 튜플 – 연산

- 덧셈

```
1  a = (1, 2, 3)
2  b = (4, 5, 6)
3  print(a+b)
```

```
>>> %Run ps.py
 (1, 2, 3, 4, 5, 6)
```

- 곱셈

```
1  a = (1, 2) * 3
2  print(a)
```

```
>>> %Run ps.py
 (1, 2, 1, 2, 1, 2)
```

# 튜플 – 순회

- 데이터 순회 & 인덱스 순회

```
1  a = (1, 2, 3)
2  for data in a:
3      print(data)
```

```
1  a = (1, 2, 3)
2  for i in range(len(a)):
3      print(a[i])
```

```
>>> %Run ps.
1
2
3
```

# 튜플 vs 리스트

- 튜플은 데이터를 수정할 수 없다.

```
1  a = (1, 2, 3)
2  a[0] = 2
```

```
>>> %Run ps.py
Traceback (most recent call last):
  File "D:\Programming\PS\ps.py", line 2, in <module>
    a[0] = 2
TypeError: 'tuple' object does not support item assignment
```

# 튜플 vs 리스트

- 같은 형태의 튜플은 같은 데이터이다.

```
>>> a = (1, 2, 3)
    b = (1, 2, 3)
    print(id(a) == id(b))

 True
```

- [1, 2, 3] 과 [1, 2, 3] 은 서로 다른 리스트 일 수 있다.
  (1, 2, 3) 과 (1, 2, 3) 은 언제나 같은 튜플이다.

# 튜플 vs 리스트

- 튜플은 리스트보다 빠르다.

```python
import time
start = time.time()
a = tuple(range(1, 1000000))
end = time.time()

print("tuple :", end - start)

start = time.time()
a = list(range(1, 1000000))
end = time.time()

print("list  :", end - start)
```

```
tuple : 0.009999036788940 43
list  : 0.019998073577880 86
```

```
tuple : 0.016002178192138672
list  : 0.018998861312866 21
```

```
tuple : 0.015002012252807617
list  : 0.021996974945068 36
```

```
tuple : 0.011999130249023438
list  : 0.019999027252197266
```

# 튜플 vs 리스트

- 튜플은 리스트보다 빠르다.
- 따라서 데이터를 수정할 일이 없다면,
  튜플을 사용하는 것이 더 효율적이다.

# 딕셔너리

- Dictionary = 사전

명사

1. 어떤 범위 안에서 쓰이는 낱말을 모아서 일정한 순서로 배열하여 싣고 그 각각의 발음, 의미, 어원, 용법 따위를 해설한 책. 최근에는 콤팩트디스크 따위와 같이 종이가 아닌 저장 매체에 내용을 담아서 만들기도 한다.

# 딕셔너리

```
{
    "key" : "value"
}
```

- 사전에서 '단어'를 기준으로 '뜻'을 찾듯
  딕셔너리는 **key**를 기준으로 **value**를 얻어낸다.
- **key**와 **value**를 **1:1** 로 **매핑**하는 자료구조

# 딕셔너리 – 생성

- **dict()** 함수 이용

```
1  a = dict()
```

- **{}** 중괄호 이용

```
3  b = {}
```

# 딕셔너리 – 생성

- 중괄호 안에 초기 데이터를 직접 넣을 수 있다.
- **key : value** 형태로 작성한다.

```
3  b = {
4      "HI-ARC": "hongik algorithm research club"
5  }
```

# 딕셔너리 – 조작

- 데이터 추가

```
1  a = dict()
2  a["key"] = "value"
3
4  print(a)
```

```
>>> %Run -c $EDITO
 {'key': 'value'}
```

- 데이터 수정

```
5  a["key"] = "new value"
6  print(a)
```

```
>>> %Run -c $EDITOR_C
 {'key': 'new value'}
```

# 딕셔너리 – 조작

- 데이터 읽기

```
8  a["key"] = "hello"
9  print(a["key"])
```

```
>>> %Run -
  hello
```

- 데이터 삭제

```
8  a["key"] = "hello"
9  b = a.pop("key")
10 print(a, b)
```

```
>>> %Run -c
  {} hello
```

```
8  a["key"] = "hello"
9  del a["key"]
10 print(a)
```

```
>>> %Run -c
  {}
```

# 딕셔너리 – 키(key)

- 딕셔너리의 key 값은 **변하지 않는 데이터**를 사용
- ex) 정수형, 실수형, 문자열, **튜플**

```
1  a = dict()
2  a[0] = 12
3  a[3.14] = "pi"
4  a["pi"] = 3.14
5
6  print(a)
```

```
>>> %Run -c $EDITOR_CONTENT
  {0: 12, 3.14: 'pi', 'pi': 3.14}
```

# 딕셔너리 – 키(key)

- 딕셔너리의 key 값은 **변하지 않는 데이터**를 사용
- 내부가 변할 수 있는 데이터 그룹은 key 가 될 수 없다.

```
6  a[list()] = "list?"
```

```
>>> %Run -c $EDITOR_CONTENT
  Traceback (most recent call last):
    File "<string>", line 6, in <module>
  TypeError: unhashable type: 'list'
```

# 딕셔너리 - 순회

- **keys()** 메서드 활용 (데이터 조회 / 수정)

```
1  a = {
2      1: 11,
3      2: 22,
4      3: 33
5  }
6
7  for key in a.keys():
8      print(key, a[key])
```

```
>>> %Run -c

 1 11
 2 22
 3 33
```

# 딕셔너리 - 순회

- **values()** 메서드 활용 (데이터 조회)

```
1  a = {
2      1: 11,
3      2: 22,
4      3: 33
5  }
6
7  for value in a.values():
8      print(value)
```

```
>>> %Run -c

  11
  22
  33
```

# 딕셔너리 - 순회

- **items()** 메서드 활용 (데이터 조회 / 수정)

```
1  a = {
2      1: 11,
3      2: 22,
4      3: 33
5  }
6
7  for key, value in a.items():
8      print(key, value)
```

```
>>> %Run -c

 1 11
 2 22
 3 33
```

# 딕셔너리 – 키 존재 확인

- **get()** 메서드 활용

```
1  a = {
2      1: 11,
3      2: 22,
4      3: 33
5  }
6
7  result1 = a.get(3)
8  result2 = a.get(4)
9
10 print(result1)
11 print(result2)
```

```
>>> %Run
33
None
```

# 딕셔너리 – 키 존재 확인

- **in** 연산자 활용

```
1  d = {
2      1: 11,
3      2: 22,
4      3: 33
5  }
6
7  key = 1
8
9  print(key in d)
```

```
>>> %Run -
  True
```

# 딕셔너리 – 메서드 정리

- 생성 : **{ }** or **dict()**
- 추가 / 조회 : **[ ]**
- 삭제 : **pop()** or **del d[key]**
- 순회 : **keys()**, **values()**, **items()**
- 키 존재 확인 : **get()**, **key in dict**

# 딕셔너리 - 연습문제

- https://www.acmicpc.net/problem/1620

나는야 포켓몬 마스터 이다솜 성공

네가 현재 가지고 있는 포켓몬 도감에서 포켓몬의 이름을 보면 포켓몬의 번호를 말하거나, 포켓몬의 번호를 보면 포켓몬의 이름을 말하는 연습을 하도록 하여라.

# 딕셔너리 - 연습문제

- 1번부터 n번 포켓몬의 이름을 저장한 뒤,
  포켓몬의 이름으로부터 번호를 얻거나,
  번호로부터 포켓몬의 이름을 알 수 있어야 한다.

- 주어지는 포켓몬의 숫자는 최대 **10만**
  질문의 숫자도 최대 **10만**

# 딕셔너리 - 연습문제

- 리스트를 사용할 수 있을까?
- i번 포켓몬의 이름을 리스트의 i번째 데이터에 저장하자.

```
1 n = int(input())
2 pocketmon = [""]
3 for i in range(n):
4     pocketmon.append(input())
```

- 그렇다면 이름으로부터 번호를 찾는 건?

# 딕셔너리 - 연습문제

- Q. 그렇다면 이름으로부터 번호를 찾는 건?
- A. index() 메서드를 사용하면 되죠!

```
6  for _ in range(m):
7      question = input()
8      if question.isnumeric():
9          print(pocketmon[int(question)])
10     else:
11         print(pocketmon.index(question))
```

# 딕셔너리 - 연습문제

- But 시간초과가 발생합니다.
  (원래는 pypy3 으로 제출해도 맞으면 안돼요..!)

| | | | | |
|---|---|---|---|---|
| 1620 | 맞았습니다!! | 118552 KB | 6352 ms | PyPy3 / 수정 |
| 1620 | 시간 초과 | | | Python 3 / 수정 |

# 시간 제한과 연산 횟수

- 모든 백준 문제에는 **‘시간 제한’** 과 **‘메모리 제한’** 이 존재

| 시간 제한 | 메모리 제한 |
| --- | --- |
| 2 초 | 256 MB |

- 알고리즘 문제를 풀 때,
  컴퓨터는 1초에 1억 번 연산할 수 있다고 가정

# 시간 제한과 연산 횟수

- 리스트의 **index(데이터)** 메서드는

  1. 리스트를 순회하면서
  2. 현재 순회 중인 데이터를 주어진 데이터와 비교하고
  3. 두 값이 서로 일치하면 현재 index 값을 알려준다.

# 시간 제한과 연산 횟수

- 극단적인 상황을 생각해보자.
- 포켓몬이 10만개 주어졌을 때,
  10만 번째 포켓몬의 번호(인덱스)를 10만 번 물어본다면?

- 반복문을 10만 x 10만 = **10억 번 돌아야** 한다.
  이론상 **10초**의 시간이 걸리므로, 시간 초과가 발생한다.

# 딕셔너리 - 연습문제

- 딕셔너리에서 **d[key]** 를 사용해 데이터를 읽는 것은 컴퓨터 입장에서 **1번의 연산**으로 취급합니다.
- 딕셔너리를 이용해서 같이 풀어봅시다.

# 딕셔너리 - 연습문제

- 정답 코드

```python
import sys
input = sys.stdin.readline

n, m = map(int, input().split())
pocketmon_info = dict()

for pocketmon_number in range(1, n+1):
    pocketmon_name = input().rstrip()

    pocketmon_info[pocketmon_name] = pocketmon_number
    pocketmon_info[pocketmon_number] = pocketmon_name

```

```python
for _ in range(m):
    question = input().rstrip()

    if question.isnumeric():
        pocketmon_number = int(question)
        print(pocketmon_info[pocketmon_number])
    else:
        pocketmon_name = question
        print(pocketmon_info[pocketmon_name])
```

# 딕셔너리 - 연습문제

- 정답 코드

```python
import sys
input = sys.stdin.readline

n, m = map(int, input().split())
pocketmon_info = dict()

for pocketmon_number in range(1, n+1):
    pocketmon_name = input().rstrip()

    pocketmon_info[pocketmon_name] = pocketmon_number
    pocketmon_info[pocketmon_number] = pocketmon_name

```

```python
for _ in range(m):
    question = input().rstrip()

    if question.isnumeric():
        pocketmon_number = int(question)
        print(pocketmon_info[pocketmon_number])
    else:
        pocketmon_name = question
        print(pocketmon_info[pocketmon_name])
```

if문 또는 else문 내부 코드가 m번 실행됨
→ 최대 **10만 번**, 따라서 1초 안에 실행가능

# 연산 횟수 계산이 어려워요..

- 지금은 어렵고 어색한 내용이 맞습니다!
  일단 **‘딕셔너리를 배웠으니 딕셔너리로 풀어보자’** 라고
  생각하고, 딕셔너리를 사용하는 것에 익숙해져 보세요.

- 연산 횟수 이야기는 다음 주에 더 자세하게 할게요!

# 쉬어가는 시간

- 컴퓨터공학과의 진로 분야

- 프로그램 개발자 (프론트엔드, 백엔드, **모바일**, 게임, PC)
- 보안 (해커, 백신 개발자)
- 임베디드 개발자 (냉장고, 세탁기, 자동차...)
- 데브옵스 (서버 컴퓨터 관리)
- 데이터분석 (파이썬을 쓰기 좋은 곳)
- 인공지능 (대학원 거의 필수....)

# 모바일

- **핸드폰, 태블릿**에서 사용하는 어플리케이션 개발

- 안드로이드 / IOS
- 크로스 플랫폼
- 웹 앱

# 모바일 – 네이티브 앱 개발

- native 라는 말 그대로, 그 운영체제에서만 실행할 수 있는 어플리케이션을 개발합니다.

- 안드로이드는 **자바 or 코틀린**
- IOS는 **Swift** 를 이용해서 개발합니다.

# 모바일 – 네이티브 앱 개발

- 운영체제에 특화되어 있는 방식으로 개발하므로 운영체제가 지원하는 다양한 기능을 사용할 수 있다.

# 모바일 – 크로스 플랫폼 앱 개발

- 플랫폼에 상관없이 사용할 수 있는 앱을 개발합니다.

- 자주 사용하는 기술
- **Flutter**
- React Native

# 모바일 – 웹 앱 개발

- **웹 기술**(html, css, javascript)을 이용해
  핸드폰 앱과 유사한 디자인의 웹 사이트를 만들고,
  이를 어플리케이션으로 만듭니다.

# 모바일 – 개인적인 추천

- **Flutter** 로 시작하는 것을 추천해요!
- 안드로이드, ios 앱을 한번에 만들 수 있습니다.
- 네이티브보다 입문하기 쉬웠어요.
  (안드로이드 기준)

# 모바일 – 공부 방법

- 인터넷 강의 (인프런, 유튜브)
- 공식 문서
- 책

# 셋 (set)

- 이름 그대로 **‘집합’** 을 의미한다.
- 중복되지 않은 데이터를 저장하는 자료구조

# 셋 (set)

- set은 **데이터가 존재하는지 확인**할 때 사용한다.
- 또는 **중복값을 제거**할 때 사용한다.

# 셋 (set) – 생성

- **set(데이터그룹)**

```
>>> s = set()
>>> s

set()
```

```
>>> a = [1, 2, 3]
>>> s = set(a)
>>> s

{1, 2, 3}
```

- **{ } 중괄호 (단, 빈 중괄호는 dict로 인식)**

```
>>> s = {1, 2, 3}
>>> type(s)

<class 'set'>
```

# 셋 (set) – 조작

- 데이터 추가

```
>>> s = set()
>>> s.add(1)
>>> s
{1}
```

- 데이터 삭제

```
>>> s = {1,2,3}
    s.remove(1)
>>> s
{2, 3}
```

- **set**에서 특정 데이터를 조회하는 방법은 없다. list로 변환해서 조회해야 한다.

# 셋 (set) – 순회

- for문 이용

```
1  s = {1,2,3}
2
3  for data in s:
4      print(data)
```

# 셋 (set) – 연산

- 수학의 집합 연산을 사용할 수 있다.

```
>>> s1 = {1,2,3}
>>> s2 = {2,3,4}
>>> s1.intersection(s2)
{2, 3}
```

교집합: intersection()
s1 & s2

```
>>> s1 = {1,2,3}
>>> s2 = {2,3,4}
>>> s1.union(s2)
{1, 2, 3, 4}
```

합집합: union()
s1 | s2

```
>>> s1 = {1,2,3}
>>> s2 = {2,3,4}
>>> s1.difference(s2)
{1}
```

차집합: difference()
(s1 – s2)

# 셋 (set) – 데이터 존재 확인

- 특정 데이터가 존재하는지 빠르게 확인할 수 있다.

```python
import time

s = set(range(1, 1000000))
l = list(range(1, 1000000))
start = time.time()
a = 999999 in s
end = time.time()

start2 = time.time()
b = 999999 in l
end2= time.time()

print("find with set: ", end - start)
print("find with list:",end2 - start2)
```

```
C:\Users\kckc0\anaconda3\python.exe D
find with set:  0.0
find with list: 0.03436231613159 18
```

리스트의 in 연산으로도 같은 작업을 할 수 있지만 걸리는 시간의 차이가 매우 큽니다.

# 셋 (set) – 중복 데이터 제거

- 중복 데이터를 제거할 수 있다.

```
group = [1, 2, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6]
s = set(group)
print(s)
```

```
C:\Users\kckc0\anaconda3
{1, 2, 3, 4, 5, 6}
```

# 셋 (set) – 메서드 정리

- 생성 : **{ x, y, ... }** or **set()**
- 추가 : **add()**
- 삭제 : **remove()**
- 순회 : **for data in s**
- 데이터 존재 확인 : **data in set**

# 셋 (set) - 연습문제

- https://www.acmicpc.net/problem/10815

숫자 카드 성공

첫째 줄에 입력으로 주어진 M개의 수에 대해서,

각 수가 적힌 숫자 카드를 상근이가 가지고 있으면 1을, 아니면 0을 공백으로 구분해 출력한다.

- 숫자 카드의 수 : 1 ~ **50만**
  각 숫자의 범위 : -1000만 ~ 1000만 → **2000만**의 숫자 범위
  질문 횟수 : 1 ~ **50만**

# 셋 (set) - 연습문제

- 리스트를 사용할 수 있을까?
- i번째 숫자 카드에 적혀 있는 수를 card[i] 에 저장하자.

```
card = list(map(int, input().split()))
```

- 그런데 만약 list에 없는 숫자만 50만 번 찾으면?
  → 50만 크기 리스트를 50만 번 돌아야 한다.
  50만 x 50만 = 2500억

# 셋 (set) - 연습문제

- 여러가지 풀이 방법이 있지만,
  집합을 사용해 같이 풀어 봅시다.

# 셋 (set) - 연습문제

- 정답 코드

```
n = int(input())
nums = set(map(int, input().split()))

m = int(input())
questions = list(map(int, input().split()))
for question_number in questions:
    print(1 if question_number in nums else 0, end=' ')
```

- print()문 코드가 낯설다면 **if문 심화** 내용을 복습해보세요!

# 셋 (set) – 연습문제

- 리스트를 다르게 사용할 수 있을까?
- 리스트의 인덱스를 숫자로 하여,
  주어진 숫자 카드의 수에 해당하는 인덱스의 값을 1로
  나머지 인덱스의 값을 0으로 설정하자.

```
1 cards = list(map(int, input().split()))
2 for number in cards:
3     having[number] = 1
```

# 셋 (set) - 연습문제

- 그런데 이렇게 저장하려면 **2000만** 범위의 숫자를 모두 저장해야 한다.

  C언어에서 정수는 4개 byte를 사용한다.
  (byte는 메모리 공간의 단위 크기)
  → 4 * 2000만 = 8000만 byte = **80MB**

# 셋 (set) - 연습문제

- 문제의 메모리 제한을 보자

| 메모리 제한 |
| --- |
| 256 MB |

- 80MB 에 비하면 매우 넉넉한 제한

# 셋 (set) - 연습문제

- 그래서 이렇게 리스트로 풀 수도 있습니다. (참고)

```python
having = [False for _ in range(20000001)]
n = int(input())
nums = list(map(int, input().split()))
for num in nums:
    having[num] = True

m = int(input())
q_list = list(map(int, input().split()))
for question in q_list:
    print(1 if having[question] else 0, end=' ')
```

# 셋 (set) - 연습문제

- 파이썬은 정수를 저장할 때 실제로는 더 많은 메모리 공간을 사용합니다.
- 보통 숫자 범위를 40억 정도로 주는 경우가 많습니다. 이 경우, 메모리 제한을 넘으니 이렇게 풀 수 없어요!

# 추가 연습 문제

- 7785 회사에 있는 사람 (https://www.acmicpc.net/problem/7785)

- 들어오고 나간 사람의 목록을 저장해야 합니다.
- 딕셔너리를 일종의 출석부처럼 사용하면 간단하게 풀 수 있어요!

# 추가 연습 문제

- 7785 회사에 있는 사람 (https://www.acmicpc.net/problem/7785)

- 들어오고 나간 사람의 목록을 저장해야 합니다.
- **딕셔너리**를 일종의 출석부처럼 사용하면 간단하게 풀 수 있어요!
- 이름이 중복되지 않기 때문에, **set**을 사용해도 됩니다!

# 추가 연습 문제

- 7785 회사에 있는 사람 (https://www.acmicpc.net/problem/7785)

- 출력할 때, 이름을 사전 순의 **'역순'** 으로 출력해야 합니다.
- 딕셔너리 또는 셋을 **sorted(), reversed() 함수**를 이용해 정렬한 뒤 뒤집어 보세요!

# 추가 연습 문제 – 딕셔너리 풀이

```python
import sys
input = sys.stdin.readline

d = dict()
n = int(input())
for _ in range(n):
    name, status = input().rstrip().split()
    # 위 코드가 헷갈리면 이렇게 쓸 수도 있어요
    # l = list(input().rstrip().split())
    # name, status = l[0], l[1]
    if status == "enter":
        d[name] = "enter"
    else:
        del d[name]

left_people = d.keys()
for name in reversed(sorted(left_people)):
    print(name)
```

# 추가 연습 문제 – 셋 풀이

```python
import sys; input=sys.stdin.readline
n = int(input())
s = set()
for _ in range(n):
    name, check = input().split()
    if check == "enter":
        s.add(name)
    else:
        s.remove(name)
for name in sorted(s, reverse=True):
    print(name)
```

# 이번주 연습문제

딕셔너리

- 10816 숫자 카드 2
- 17219 비밀번호 찾기
- 25192 인사성 밝은 곰곰이
- 28446 볼링공 찾아주기

셋

- 14425 문자열 집합
- 1764 듣보잡
- 1269 대칭 차집합