5. 파이썬 프로그래밍 기초3
- 다차원 자료구조, 사전 -

자료 구조란?

- 문제 해결을 위해 많은 데이터를 효율적으로 저장하고, 처리하고, 관리할 수 있도록 자료들 사이의 관계를 나타내는 것.

- 많은 양의 데이터를 효율적으로 저장 및 처리 하려면 그에 적합한 자료 구조 설계 및 활용 능력이 필요함.

리스트

- 일련의 여러 값들을 유연하게 다루는 자료구조
  - 인덱스를 사용하여 개별 요소 접근 가능
  - 데이터 추가, 삭제가 용이 (전체 크기/길이 도 자유롭게 변경 가능)
  - 리스트 각 요소 데이터 타입이 다를 수 있도록 허용
  - 정렬, 검색, 수정과 같은 편리한 기능을 함께 제공

다차원 리스트

- 리스트 속의 리스트 (중첩 리스트) : 복잡한 자료 구조도 쉽게 표현할 수 있음.

```
# 학생을 표현하기 위해 어떤 정보들이 필요할까?
student1 = [ 2022001, "신사임당", ['소속대학1', '소속학과1'], ['A', 'B+', 'A+', 'A', 'A+'] ]
student2 = [ 2022002, "홍길동", ['소속대학2', '소속학과2'], ['A', 'A+', 'B+', 'A', 'A+'] ]
print( student1 )
print( student1[1] )
print( student1[3] )
grade = student1[3]
print( grade[0] )
print( student1[3][0] )
```

```
[2022001, '신사임당', ['소속대학1', '소속학과1'], ['A', 'B+', 'A+', 'A', 'A+']]
신사임당
['A', 'B+', 'A+', 'A', 'A+']
A
A
```

다차원 리스트

다차원 리스트 원소/데이터 접근 하는 방법

```
# 학생을 표현하기 위해 어떤 정보들이 필요할까?
student1 = [ 2022001, "신사임당", ['소속대학1', '소속학과1'], ['A', 'B+', 'A+', 'A', 'A+'] ]
student2 = [ 2022002, "홍길동", ['소속대학2', '소속학과2'], ['A', 'A+', 'B+', 'A', 'A+'] ]
student_list = [student1, student2]
print( student_list )
print( student_list[0] )
print( student_list[0][3] )
print( student_list[0][3][1] )
```

```
[
  [2022001, '신사임당', ['소속대학1', '소속학과1'], ['A', 'B+', 'A+', 'A', 'A+']],
  [2022002, '홍길동', ['소속대학2', '소속학과2'], ['A', 'A+', 'B+', 'A', 'A+']]
]
[2022001, '신사임당', ['소속대학1', '소속학과1'], ['A', 'B+', 'A+', 'A', 'A+']]
['A', 'B+', 'A+', 'A', 'A+']
'B+'
```

예제: 다차원 리스트 기반 성적 관리

```
class_grades = list()
num_students = int( input("학생 수 입력: ") )
num_scores = int( input("과목 수 입력: ") )

for i in range(num_students):
        tmp_list = list()
        print(i+1, "번째 입니다")
        name = input("학생 이름을 입력하세요: ")
        tmp_list.append(name)
        print("개별 과목 점수를 입력합니다.")

        for j in range(num_scores):
                score = input("점수 입력: ")
                tmp_list.append(score)

        class_grades.append(tmp_list)

print(class_grades)
```

```
학생 수 입력: 2
과목 수 입력: 3
1 번째 입니다
학생 이름을 입력하세요: 신사임당
개별 과목 점수를 입력합니다.
점수 입력: 100
점수 입력: 90
점수 입력: 95
2 번째 입니다
학생 이름을 입력하세요: 홍길동
개별 과목 점수를 입력합니다.
점수 입력: 95
점수 입력: 95
점수 입력: 100
[['신사임당', 100, 90, '95'],['홍길동', '95', '95', '100']]
```

예제: 다차원 리스트 기반 성적 관리

```
print(class_grades[0])
print(class_grades[0][0])
print(class_grades[0][1:])
st_name = class_grades[0][0]
score_cnt = len(class_grades[0][1:])
score_sum = sum(class_grades[0][1:])

print(st_name, "평균점수:", score _sum/score_cnt)
```

```
['신사임당', '100', '90', '95']
신사임당
['100', '90', '95']
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    score_sum = sum(class_grades[0][1:])
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

성적 점수가 문자열 타입으로 저장되어 있음.
sum( ) 함수를 사용하려면 ?

예제: 다차원 리스트 기반 성적 관리

```
print(class_grades[0])
print(class_grades[0][0])
print(class_grades[0][1:])
st_name = class_grades[0][0]

scores = class_grades[0][1:]
score_cnt = len(scores)
for i in range(score_cnt):   # 성적 점수 타입을 정수형 타입으로 변환 저장
        scores[i] = int(scores[i])
score_sum = sum(scores)

print(st_name, "평균점수:", score _sum/ score_cnt)
```

```
['신사임당', '100', '90', '95']
신사임당
['100', '90', '95']
신사임당 평균점수: 95.0
```

사전 (Dictionary)

- 키(key)와 값(value)의 쌍으로 구성됨
  - 변수이름 = { key1 : value1, key2 : value2 ... }
- 사용 예시
  - 식당에서 선택한 음식의 가격을 확인하고 싶을 때
  - 영한 사전에서 특정 영어 단어의 뜻을 알고 싶을 때
  - 학번으로 학생 정보를 조회하고 싶을 때

```
menu = {"햄버거" : 4000, "파스타" : 9000}
word_dic = {"apple" : "사과", "banana" : "바나나"}
students = {2022001 : "신사임당", 2022002 : "정약용"}
```

리스트와 사전의 비교

- 리스트는 각 원소를 접근하기 위해서 **인덱스를 사용**

```
L = ['신사임당', '정약용', '장영실', '이순신']
print ( L [ 3 ] )
```

신사임당	정약용	장영실	이순신
0	1	2	3

- 사전은 각 원소를 **접근하기 위해서 키 (key) 를 사용**

```
D = { 2022001:'신사임당', 2022002:'정약용', 2022003:'장영실', 2022004:'이순신'}
print( D[ 2022004 ] )
```

key	value
2022001	신사임당
2022002	정약용
2022003	장영실
2022004	이순신

사전에 원소의 추가 또는 삭제

- 사전에 새로운 원소를 추가하는 방법
  - dictionary[key] = value

    # 새로운 key와 value 의 쌍을 추가
  - 사전은 리스트와 달리 인덱스 개념이 없음

- 사전에 존재하는 원소를 삭제하는 방법
  - del dictionary[key]

    # key에 해당하는 원소의 쌍을 삭제

```
student1 = {'학번':2022001, '이름':'이순신', '학과':'불멸학과'}
print(student1)
print(student1['학번'], student1['이름'], student1['학과'])
print()
student1['연락처'] = '010-1234-1234'
print(student1)
print(student1['이름'], student1['연락처'])
print()
del student1['연락처']
print(student1)
```

```
{'학번': 2022001, '학과': '불멸의학과', '이름': '이순신'}
2022001 이순신 불멸의학과

{'학번': 2022001, '학과': '불멸의학과', '이름': '이순신', '연락처': '010-1234-1234'}
이순신 010-1234-1234

{'학번': 2022001, '학과': '불멸의학과', '이름': '이순신'}
```

예제: 편의점 재고 관리

- 편의점에서 재고 관리를 수행하는 프로그램을 작성해보자.

편의점에서 판매하는 물건의 재고를 사전 에 저장한다.

```
products = { "커피캔": 7, "펜": 3, "종이컵": 2, "우유": 1, "콜라": 4, "책": 5 }


while True:
          item = input("재고 확인할 물건은? ")
          print (products[item])
```

```
재고 확인할 물건은? 콜라
4
재고 확인할 물건은? 종이컵
2
재고 확인할 물건은? 우유
1
재고 확인할 물건은? 커피캔
7
```

리스트 vs. 사전

- 다차원 리스트로 유사한 표현이 가능함, 그러나 리스트에서는 특정 항목을 찾기 위해선 순차적 검색이 필요함.

```
products = [ ["커피캔", 7], ["펜", 3], ["종이컵", 2], ["우유", 1], ["콜라", 4], ["책", 5] ]
while True:
    item = input("재고 확인할 물건은? ")
    for tmp in products:
        if item == tmp[0]:
            print(tmp[1])
            break
```

- 사전은 key 를 사용하여 단 번의 특정 항목을 찾을 수 있음.

```
products = { "커피캔": 7, "펜": 3, "종이컵": 2, "우유": 1, "콜라": 4, "책": 5 }
while True:
          item = input("재고 확인할 물건은? ")
          print (products[item])
```

사전의 value 로 리스트 타입 사용 예시

```
students = dict()  # 빈 사전을 선언하는 법
students[2022001] = ['이순신', '군사전략학과']
students[2022002] = ['잭바우어', '경호학과']
print(students[2022001])
st = students[2022001]
print(st[0], st[1])
print(students[2022001][0], students[2022001][1])
print()
print(students[2022002])
st = students[2022002]
print(st[0], st[1])
print(students[2022002][0], students[2022002][1])
```

```
['이순신', '군사전략학과']
이순신 군사전략학과
이순신 군사전략학과

['잭바우어', '경호학과']
잭바우어 경호학과
잭바우어 경호학과
```

사전의 value 로 사전 타입 사용 예시

```
students = dict()  # 빈 사전을 선언하는 법
students[2022001] = {'이름':'이순신', '학과':'군사전략학과'}
students[2022002] = {'이름':'잭바우어', '학과':'경호학과'}
print(students[2022001])
st = students[2022001]
print(st['이름'], st['학과'])
print(students[2022001]['이름'], students[2022001]['학과'])
print()
print(students[2022002])
st = students[2022002]
print(st['이름'], st['학과'])
print(students[2022002]['이름'], students[2022002]['학과'])
```

```
{'이름': '이순신', '학과': '군사전략학과'}
이순신 군사전략학과
이순신 군사전략학과

{'이름': '잭바우어', '학과': '경호학과'}
잭바우어 경호학과
잭바우어 경호학과
```

사전 & 반복 문

- 사전은 인덱스 개념을 적용할 수 없음. for 반복 문을 사용하면 key 를 하나씩 가져올 수 있음.
- key 를 안다면 key 에 연관된 value 를 알 수 있음.

```
foods = {'떡볶이':'튀김', '짜장면':'단무지', '라면':'김치'}
for key in foods:
        print(key,"은", foods[key], "와 제법 어울려요!")
print()
foods['치킨'] = '맥주'  # 새로운 치킨 & 맥주 를 추가함
for key in foods:
        print(key,"은", foods[key], "와 제법 어울려요!")
```

```
떡볶이 은 튀김 와 제법 어울려요!
짜장면 은 단무지 와 제법 어울려요!
라면 은 김치 와 제법 어울려요!

떡볶이 은 튀김 와 제법 어울려요!
짜장면 은 단무지 와 제법 어울려요!
라면 은 김치 와 제법 어울려요!
치킨 은 맥주 와 제법 어울려요!
```

그 외 사전 활용 팁

- 파이썬 사전은 중복 key 를 가질 수 없음.
  - 이미 존재하는 key 에 대해 다른 value 를 저장하면 덮어써 짐.

```
foods = {'떡볶이':'튀김', '라면':'김치', '피자':'피클'}

print(foods)
foods['떡볶이'] = '순대'
print(foods)
```

  - 존재하지 않은 key 로 접근하면 에러가 발생하므로, 미리 확인이 필요함.

```
foods = {'떡볶이':'튀김', '라면':'김치', '피자':'피클'}

while True:
        print('~~~~~~~~~~~~~~~~~~~~~~~~')
        food1 = input('음식 명: ')
        if food1 in foods:
                print(food1 + '에는' + ' ' + foods[food1] + '가 어울리죠!' )
        else:
                food2 = input(food1 + '에 어울리는 음식은 무엇인가요? ')
                foods[food1] = food2
```

```
~~~~~~~~~~~~~~~~~~~~~~~~
음식 명: 치킨
치킨에 어울리는 음식은 무엇인가요?
맥주
~~~~~~~~~~~~~~~~~~~~~~~~
음식 명: 치킨
치킨에는 맥주가 어울리죠!
~~~~~~~~~~~~~~~~~~~~~~~~
```

사전을 리스트로 변환 하기

```
foods = {'떡볶이':'튀김', '라면':'김치', '피자':'피클'}

foods_keys = foods.keys()
foods_values = foods.values()
foods_items = foods.items()

print(foods_keys)
print(foods_values)
print(foods_items)

print('~~~~~~~~~~~~~~~~~~~~~~~~~')
print('After converting to lists')
print('~~~~~~~~~~~~~~~~~~~~~~~~~')

to_list = list( foods_keys )
print(to_list)

to_list = list(foods_values )
print(to_list)

to_list = list(foods_items )
print(to_list)
```

```
dict_keys(['떡볶이', '피자', '라면'])
dict_values(['튀김', '피클', '김치'])
dict_items([('떡볶이', '튀김'), ('피자', '피클'), ('라면', '김치')])
~~~~~~~~~~~~~~~~~~~~~~~~~
After converting to lists
~~~~~~~~~~~~~~~~~~~~~~~~~
['떡볶이', '피자', '라면']
['튀김', '피클', '김치']
[('떡볶이', '튀김'), ('피자', '피클'), ('라면', '김치')]
```

자료 구조 리뷰

- 데이터를 효율적으로 저장하고 꺼내는 방법론
- 사용하는 자료구조에 따라 구현과정/성능에 큰 영향
  - 단일 변수, 리스트, 사전 등

```
st_list = [ [2022001, "Mike"], [202002, "Jhon"] ]
key = int(input("ID: "))
for st in st_list:
        if key == st[0]:
                print(st[0], st[1])
```

```
st_dict = { 2022001:'Mike', 2022002:'Jhon' }
key = int(input("ID: "))
print (key, st_dict[key])
```

자료 구조와 알고리즘 관계

- 동일한 목표에 대해서도 어떤 자료구조를 사용했는지에 따라 사용하기 적합한 알고리즘이 달라 질 수 있음.

- 연속적인 숫자 입력을 받아 최대값을 구하는 경우, 단일 변수, 리스트, 사전 중 어느 것을 사용하는 것이 가장 좋을까?

단일 변수 vs. 리스트

```
max_num = int(input("enter num: "))
for i in range(5):
        new_num = int(input("enter num: "))
        if new_num > max_num :
            max_num = new_num
print(max_num)
```

매번 입력 받은 숫자 값을 현재까지 알고 있는 최대값과 비교하여 업데이트

```
num_list = list()
for i in range(5):
        new_num = int(input("enter num: "))
        num_list.append(new_num)

num_list.sort()
print( num_list[ len(num_list)-1 ] )
```

모든 입력 받는 숫자를 리스트에 저장한 후, 정렬하여 마지막에 위치한 값을 출력

요구사항에 따른 자료구조 선택

- 입력 숫자가 매우 많지만 최대 값 하나만 필요하면?
- 두 번째로 큰 값이 알고 싶다면 ?
- 최대 값이 몇 번째 입력이었었는지 알고 싶다면?
- 각 숫자가 몇 번째 입력이었었는지 알고 싶다면 ?
- 동일한 숫자가 몇 번씩 출현하였는지 카운팅 하려면?
- 입력된 수 중 가장 높은 10개 숫자는 ?
- 최근 10개 중 가장 최대 값은 ?

예제: 동일한 데이터/정보 카운트하기

- 책, 웹 페이지, 설문조사 등의 데이터에서 특정 단어/키워드 출현 빈도는 중요한 의미를 가짐.
  - 워드 카운터 (예: 1000 자 이내로 작성)
  - SNS 상에서 특정 인물이 거론되는 횟수
  - 사용되는 단어 빈도수로 감정 분석
  - 보행자/고객의 이동 경로 분석

- 정보를 얻을 때마다 이미 출현한 적이 있다면 정보를 갱신하고, 아니면 새로 추가한다.

예제: 동일한 데이터/정보 카운트하기

```
import random

num_loops = 10000
max_num = 1000

for i in range(num_loops):                    # num_loops 만큼 반복 정보

    RN = random.randint(0, max_num)           # max_num 이하 랜덤 수 생성

    print(RN)
```

매번 랜덤 하게 생성한 숫자와 그 숫자의 출현 빈도수를 어떻게 기록할 것인가?

예제: 동일한 데이터/정보 카운트하기

- 리스트를 사용하면 각 정보에 대해 [정보, 출현횟수] 형태로 추가 / 업데이트

```
import random
num_loops = 10000
max_num = 1000
nums_list = list()
for i in range(num_loops):
    RN = random.randint(0, max_num)
    notFound = True
    for j in range( len(nums_list) ):
        if RN == nums_list[j][0]:   # RN 값이 이미 존재하는지 확인
            nums_list[j][1] = nums_list[j][1] + 1
            notFound = False
            break
    if notFound: # 첫 출현이라면 [RN, 1] 형태로 추가
        nums_list.append( [RN, 1] )
```

예제: 동일한 데이터/정보 카운트하기

- 사전을 사용하면 각 정보에 대해 {정보 : 출현횟수} 형태로 추가 / 업데이트

```
import random

num_loops = 10000
max_num = 1000
nums_dict = dict()
for i in range(num_loops):
    RN = random.randint(0, max_num)
    if RN in nums_dict:    # RN 값이 이미 존재하는지 확인
        nums_dict[RN] = nums_dict[RN] + 1
    else:                  # 첫 출현이라면 {RN:1} 형태로 저장
        nums_dict[RN] = 1
```

* 코드 라인 수가 적을 뿐만 아니라, 수행 시간에서 큰 차이가 발생