Data Structure 02

딕셔너리 주요 함수

- keys() : 딕셔너리의 모든 키를 반환
- values() : 딕셔너리의 모든 값을 반환
- items() : 딕셔너리의 모든 키와 값을 튜플로 반환
- get(key) : 지정된 키의 값을 반환
- update(dict) : 다른 딕셔너리의 내용을 병합
- pop(key) : 지정된 키의 값을 제거하고 해당 값을 반환하며 키가 없으면 None을 반환
- clear() : 딕셔너리의 모든 요소를 제거

딕셔너리 주요 함수

```
# 키를 반환
dict = {"a": 1, "b": 2, "c": 3}
print(dict.keys())
# 출력: dict_keys(['a', 'b', 'c'])

# 값을 반환
print(dict.values())
# 출력: dict_values([1, 2, 3])

# 키와 값을 튜플로 반환
print(dict.items())
# 출력: dict_items([('a', 1), ('b', 2), ('c', 3)])

# 지정된 키의 값을 반환
print(dict.get("a"))
# 출력: 1
```

딕셔너리 주요 함수

```
# 지정된 키의 값을 설정
dict["d"] = 4
print(dict)
# 출력: {'a': 1, 'b': 2, 'c': 3, 'd': 4}

# 지정된 키의 값을 제거하고 해당 값을 반환
print(dict.pop("a"))
# 출력: 1
print(dict)
# 출력: {'b': 2, 'c': 3, 'd': 4}

# 딕셔너리의 모든 요소를 제거
dict.clear()
print(dict)
# 출력: {}
```

딕셔너리 – 예제

```
# 트렌드 코리아 2024 단어장 초기 설정

trends = {
    "분초사회": "시간은 중요한 자원이며, 효율적인 시간 사용을 위한 빠르고 편리한 소비 경험을 제공하는 기업이 유리하다.",
    "호모 프롬프트": "AI 발전으로 다양한 정보와 문제 해결이 가능해졌으며, 사용자는 적절한 답변을 제공하는 AI를 선호한다.",
    "도파밍": "재미와 자극적인 콘텐츠를 추구하는 소비자가 증가하므로, 기업은 새로운 재미를 제공해야 한다.",
    "요즘남편 없던아빠": "남성의 가사 노동과 육아 참여가 증가하므로, 남성 소비자의 니즈를 반영한 제품과 서비스가 필요하다.",
    "디토소비": "복잡한 구매 과정을 피하려는 소비자가 증가하므로, 소비자의 취향을 반영한 제품과 서비스가 필요하다.",
    "리퀴드폴리탄": "유목적 라이프스타일이 확산되면서 지역의 개념이 변화하므로, 지역을 넘어서는 전략이 필요하다.",
    "돌봄경제": "돌봄의 필요성이 증가하므로, 돌봄 서비스 시장이 성장하며, 기업은 이 산업에 적극 참여해야 한다."
}
```

딕셔너리 – 예제

```
# 전체 목록 보기 함수
def view_trends():
    for key, value in trends.items():
        print(f"{key} : {value}")

# 단어 추가 함수
def add_trend():
    key = input("추가할 트렌드 이름: ")
    value = input("트렌드에 대한 설명: ")
    trends[key] = value
```

딕셔너리 – 예제

```
# 단어 삭제 함수
def delete_trend():
    key = input("삭제할 트렌드 이름: ")
    if key in trends:
        del trends[key]
    else:
        print("해당 트렌드는 존재하지 않습니다.")

# 단어 검색 함수
def search_trend():
    key = input("검색할 트렌드 이름: ")
    if key in trends:
        print(f"{key} : {trends[key]}")
    else:
        print("해당 트렌드는 존재하지 않습니다.")
```

```
def main():
    while True:
        print("\n 트렌드 단어장 메뉴 ---")
        print("1. 전체 목록 보기")
        print("2. 단어 추가")
        print("3. 단어 삭제")
        print("4. 단어 검색")
        print("5. 종료")
        choice = int(input("선택: "))
        if choice == 1:
            view_trends()
        elif choice == 2:
            add_trend()
        elif choice == 3:
            delete_trend()
        elif choice == 4:
            search_trend()
        elif choice == 5:
            print("프로그램을 종료합니다.")
            break
        else:
            print("잘못된 선택입니다. 다시 입력해주세요.")
main()
```

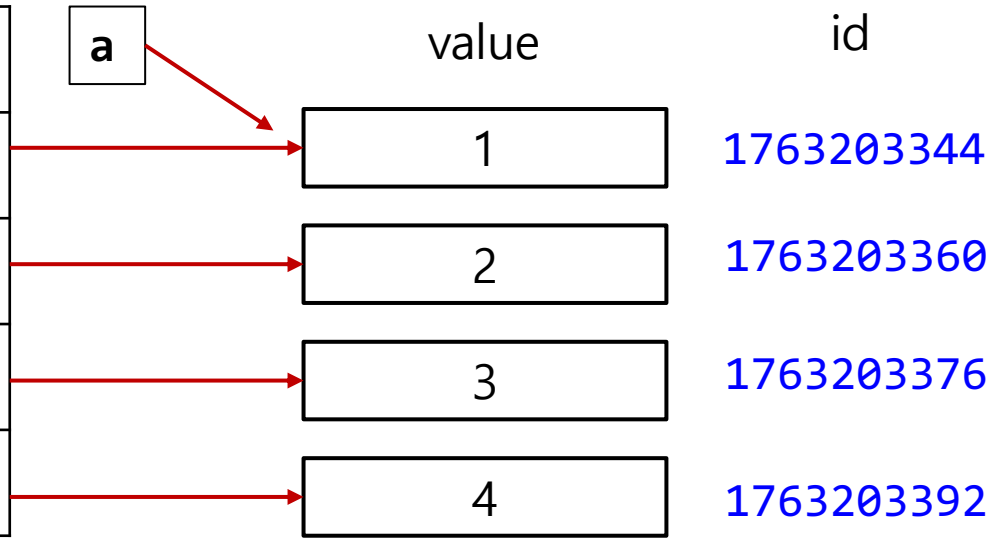
```
 트렌트 단어장 메뉴 ---
1. 전체 목록 보기
2. 단어 추가
3. 단어 삭제
4. 단어 검색
5. 종료
선택: 4
검색할 트렌드 이름: 분초사회
분초사회 : 시간은 중요한 자원이며, 효율적인 시간 사용을 위한 빠르고 편리한 소비 경험을 제공하는 기업이 유리하다.

 트렌트 단어장 메뉴 ---
1. 전체 목록 보기
2. 단어 추가
3. 단어 삭제
4. 단어 검색
5. 종료
선택: 3
삭제할 트렌드 이름: 분초사회

 트렌트 단어장 메뉴 ---
1. 전체 목록 보기
2. 단어 추가
3. 단어 삭제
4. 단어 검색
5. 종료
선택: 1
호모 프롬프트 : AI 발전으로 다양한 정보와 문제 해결이 가능해졌으며, 사용자는 적절한 답변을 제공하는 AI를 선호한다.
육각형인간 : 다양한 분야에서 뛰어난 성과를 추구하는 소비자가 증가하므로, 기업은 다양한 욕구를 충족시키는 제품과 서비스
가 필요하다.
버라이어티 가격 전략 : 빅데이터와 AI를 통해 소비자의 지불 의향과 시장 상황을 파악하여 다양한 가격 전략을 적용한다.
도파밍 : 재미와 자극적인 콘텐츠를 추구하는 소비자가 증가하므로, 기업은 새로운 재미를 제공해야 한다.
요즘남편 없던아빠 : 남성의 가사 노동과 육아 참여가 증가하므로, 남성 소비자의 니즈를 반영한 제품과 서비스가 필요하다.
스핀오프 프로젝트 : 새로운 비즈니스 기회를 찾는 기업이 늘어나므로, 유망한 아이디어를 발굴하고 실험하는 것이 중요하다.
디토소비 : 복잡한 구매 과정을 피하려는 소비자가 증가하므로, 소비자의 취향을 반영한 제품과 서비스가 필요하다.
리퀴드폴리탄 : 유목적 라이프스타일이 확산되면서 지역의 개념이 변화하므로, 지역을 넘어서는 전략이 필요하다.
돌봄경제 : 돌봄의 필요성이 증가하므로, 돌봄 서비스 시장이 성장하며, 기업은 이 산업에 적극 참여해야 한다.
```

리스트의 메모리 저장

```
list1=[1,2,3,4]
print(list1)
print(id(list1[0]), id(list1[1]), id(list1[2]),id(list1[3]))
a=1
print(id(a))
```

```
[1, 2, 3, 4]
1763203344 1763203360 1763203376 1763203392
1763203344
```

index	list1
0	1763203344
1	1763203360
2	1763203376
3	1763203392



리스트 – 패킹과 언패킹

```
p = [1,2,3]          # 1,2,3을 변수 p에 패킹
a,b,c = p            # p에 있는 값 1,2,3을 변수 a,b,c 에 언패킹
print(p, a,b,c)
```

```
[1,2,3] 1 2 3
```

이차원 리스트

학생	A	B	C	D	E
국어	80	75	97	77	85
수학	65	95	88	78	95
영어	75	88	90	97	90

```
kor=[80,75,97,77,85]
math=[65,95,88,78,95]
eng=[75,88,90,97,90]

mid=[kor,math,eng]
print(mid)
```

```
[[80, 75, 97, 77, 85], [65, 95, 88, 78, 95], [75, 88, 90, 97, 90]]
```

이차원 리스트

mid	학생	A	B	C	D	E
mid[0]	kor	80 kor[0] mid[0][0]	75 kor[1] mid[0][1]	97 kor[2] mid[0][2]	77 kor[3] mid[0][3]	85 kor[4] mid[0][4]
mid[1]	math	65 math[0] mid[1][0]	95 math[1] mid[1][1]	88 math[2] mid[1][2]	78 math[3] mid[1][3]	95 math[4] mid[1][4]
mid[2]	eng	75 eng[0] mid[2][0]	88 eng[1] mid[2][1]	90 eng[2] mid[2][2]	97 eng[3] mid[2][3]	90 eng[4] mid[2][4]

이차원 리스트

```
kor=[80,75,97,77,85]
math=[65,95,88,78,95]
eng=[75,88,90,97,90]

mid=[kor,math,eng]
print("mid[0] = ", mid[0])
print("mid[1] = ", mid[1])
print("mid[2] = ", mid[2])

mid[0]=100

print("mid = ", mid)
```

```
mid[0] =  [80, 75, 97, 77, 85]
mid[1] =  [65, 95, 88, 78, 95]
mid[2] =  [75, 88, 90, 97, 90]
mid =  [100, [65, 95, 88, 78, 95], [75, 88, 90, 97, 90]]
```

이차원 리스트

```
kor=[80,75,97,77,85]
math=[65,95,88,78,95]
eng=[75,88,90,97,90]

mid=[kor,math,eng]
print("mid[0] = ", mid[0])
print("mid[1] = ", mid[1])
print("mid[2] = ", mid[2])

mid[0][0]=100

print("mid = ", mid)
```

```
mid[0] =  [80, 75, 97, 77, 85]
mid[1] =  [65, 95, 88, 78, 95]
mid[2] =  [75, 88, 90, 97, 90]
mid =  [[100, 75, 97, 77, 85], [65, 95, 88, 78, 95], [75, 88, 90, 97, 90]]
```

이차원 리스트

```
kor=[80,75,97,77,85]
math=[65,95,88,78,95]
eng=[75,88,90,97,90]

mid=[kor,math,eng]
print("mid[0] = ", mid[0])
print("mid[1] = ", mid[1])
print("mid[2] = ", mid[2])

print("eng[4] = ", eng[4])
print("mid[2][4] = ", mid[2][4])
```

```
mid[0] =  [80, 75, 97, 77, 85]
mid[1] =  [65, 95, 88, 78, 95]
mid[2] =  [75, 88, 90, 97, 90]
eng[4] =  90
mid[2][4] =  90
```

리스트 컴프리헨션(list comprehension)

리스트 컴프리헨션(List Comprehension)은 리스트를 생성하기 위한 간결하고 표현력이 높은 문법

```
[expression for item in iterable]
```

expression : 현재의 item에 대한 연산이나 처리 결과

item : iterable에서 현재 항목을 나타냄

iterable : 순회 가능한 객체(예: 리스트, 튜플, 문자열 등)

리스트 컴프리헨션(list comprehension)

```
x=[]
for i in range(10):
      x.append(i)

print("x=", x)
```

```
x= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
# list comprehension: 기존 리스트형을 사용하여 간단하게 새로운 리스트를 만드는 기법

y=[i for i in range(10)]

print("y=", y)
```

```
y= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

리스트 컴프리헨션(list comprehension)-필터링

```
x=[]
for i in range(10):
      if i % 2 == 0:
            x.append(i)

print("x=", x)
```

```
x= [0, 2, 4, 6, 8]
```

```
y=[i for i in range(10) if i % 2 == 0] #짝수만 저장

print("y=", y)
```

```
y= [0, 2, 4, 6, 8]
```

리스트 컴프리헨션(list comprehension)-필터링

```
#else 사용하여 조건을 만족하지 않을 시 다른 값 할당
y=[i if i % 2 == 0 else 99 for i in range(10)]

print("y=", y)
```

```
y= [0, 99, 2, 99, 4, 99, 6, 99, 8, 99]
```

리스트 컴프리헨션(list comprehension)-nested loop

```
w1="Hello"
w2="World“

# w1에 나오는 값을 먼저 고정한 후 w2의 값을 하나씩 가져와 결과 생성
r=[i+j for i in w1 for j in w2] # nested loop

print("r = ", r)
```

```
r =  ['HW', 'Ho', 'Hr', 'Hl', 'Hd', 'eW', 'eo', 'er', 'el', 'ed', 'lW',
'lo', 'lr', 'll', 'ld', 'lW', 'lo', 'lr', 'll', 'ld', 'oW', 'oo', 'or',
'ol', 'od']
```

리스트 컴프리헨션(list comprehension)-nested loop

```
c1=["A","B","C"]
c2=["D","E","A"]

r=[i+j for i in c1 for j in c2 if not (i==j)] # i == j 필터링

print("r = ", r)
```

```
r =  ['AD', 'AE', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
```

리스트 컴프리헨션(list comprehension)-nested loop

```
[expression for outer_loop_variable in outer_iterable for inner_loop_variable
in inner_iterable]
```

```
# 2단부터 9단까지의 결과를 리스트 컴프리헨션을 사용하여 생성
gugudan = [(a, b, a*b) for a in range(2, 10) for b in range(1, 10)]

# 결과 출력
for item in gugudan:
    print(f"{item[0]} x {item[1]} = {item[2]}")
```

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
```

리스트 컴프리헨션(list comprehension)-이차원 리스트

```
ws="Anyone can be anything".split()
print("ws=", ws)

#리스트의 각 요소를 대문자,소문자, 길이로 변환하여 이차원 리스트로 변환
stuff=[[w.upper(), w.lower(), len(w)]for w in ws]

print("stuff=", stuff)

for i in stuff:
      print(i)
```

```
ws= ['Anyone', 'can', 'be', 'anything']

stuff= [['ANYONE', 'anyone', 6], ['CAN', 'can', 3], ['BE', 'be', 2], ['ANYTHING', 'anything', 8]]

['ANYONE', 'anyone', 6]
['CAN', 'can', 3]
['BE', 'be', 2]
['ANYTHING', 'anything', 8]
```

리스트 컴프리헨션(list comprehension)-이차원 리스트

```
c1=["A","B","C"]
c2=["D","E","A"]

r1=[i+j for i in c1 for j in c2] # 1차원 리스트, 앞 for 먼저 실행
print("r1 = ", r1)


r2=[[i+j for i in c1] for j in c2] # 2차원 리스트, 뒤 for 먼저 실행
print("r2 = ", r2)
```

```
r1 =  ['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']

r2 =  [['AD', 'BD', 'CD'], ['AE', 'BE', 'CE'], ['AA', 'BA', 'CA']]
```

디폴트 인수(default arguments)

```
def da_test(a,b,c=10,d=20):
      sum=a+b+c+d
      return(sum)

print(da_test(50,60)) #Output: 140

print(da_test(10,20,30,40)) # Output: 100




def da_test(a,b=10,c,d=20): # error
      sum=a+b+c+d
      return(sum)

print(da_test(10,20,30,40))
```

가변 인수(variable length arguments)

- ***args**는 함수에 임의의 개수의 위치 인자를 전달할 수 있게 해줌
- 인자들은 함수 내부에서 **튜플**로 처리

```
def print_args(*args):
    for arg in args:
        print(arg)

print_args(1, 2, 3, 4)
```

```
# 출력: 1 2 3 4
```

키워드 가변 인수(keyword variable length arguments)

- **kwargs는 함수에 임의의 개수의 키워드 인자를 전달할 수 있게 해줌
- 인자들은 함수 내부에서 딕셔너리로 처리

```
def print_kwargs(**kwargs):
    for key, value in kwargs.items():
        print(f"{key} = {value}")

print_kwargs(a=1, b=2, c=3)
```

```
# 출력: a = 1, b = 2, c = 3
```

스택(stack)-LIFO(List in Fist Out)

```
stack = []
stack.append('a')
stack.append('b')
stack.append('c')
print('Initial stack:', stack)

print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:', stack)
```

```
Initial stack: ['a', 'b', 'c']

Elements popped from stack:
c
b
a

Stack after elements are popped: []
```

큐(queue)-FIFO(Fist in Fist Out)

```
queue = []
queue.append('a')
queue.append('b')
queue.append('c')
print('Initial queue:', queue)

print('\nElements dequeued from queue:')
print(queue.pop(0))
print(queue.pop(0))
print(queue.pop(0))

print('\nQueue after elements are dequeued:', queue)
```

```
Initial queue: ['a', 'b', 'c']

Elements dequeued from queue:
a
b
c

Queue after elements are dequeued: []
```

집합(Set)

Set – 순서없이 저장, 데이터의 중복을 불허

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

print("s = ", s)

ss = {10,20,30,40,50,10,20,30}

print("\ns = ", ss)
```

```
s =  {1, 2, 3, 4, 5}

s =  {50, 20, 40, 10, 30}
```

Set example 1

```
my_set = set()

my_set.add(1)
my_set.add(2)
my_set.add(3)

print(my_set) # Output: {1, 2, 3}

my_set.update([1,3,4,5,6,7])

print(my_set) # Output: {1, 2, 3, 4, 5, 6, 7}

my_set2 = {10, 20, 30}

my_set2.add(40)
my_set2.add(10)
my_set2.add(50)

my_set2.remove(20)

print(my_set2) # Output: {40, 10, 50, 30}
```

Set example 2

```
set1 = {1, 2, 3}
set2 = {2, 3, 4}

# 합집합(Union)
print(set1.union(set2)) # Output: {1, 2, 3, 4}
print(set1 | set2)      # Output: {1, 2, 3, 4}

# 교집합(Intersection)
print(set1.intersection(set2)) # Output: {2, 3}
print(set1 & set2)             # Output: {2, 3}

# 차집합(Difference)
print(set1.difference(set2)) # Output: {1}
print(set1 - set2)           # Output: {1}
```

collections 모듈

- collections – 파이썬 내장 모듈(build-in data structure)

  deque : 양쪽 끝에서 삽입과 삭제 가능, 스택과 큐를 모두 지원

```
from collections import deque

queue = deque([1, 2, 3])
print(queue)
queue.append(4)
print(queue)
queue.append(5)
print(queue)
queue.popleft()
print(queue)
queue.pop()
print(queue)
```

```
deque([1, 2, 3])
deque([1, 2, 3, 4])
deque([1, 2, 3, 4, 5])
deque([2, 3, 4, 5])
deque([2, 3, 4])
```

collections 모듈

- collections – 파이썬 내장 모듈(build-in data structure)

  OrderedDict : 입력된 순서로 key-value 저장, key 또는 value로 데이터 정렬 가능

```
from collections import OrderedDict

d = OrderedDict()
d['b'] = 100
d['a'] = 700
d['d'] = 300
d['c'] = 500
d['e'] = 200

for key, value in d.items():
    print(key, value)
```

```
b 100
a 700
d 300
c 500
e 200
```

collections 모듈

- collections – 파이썬 내장 모듈(build-in data structure)

OrderedDict : 입력된 순서로 key-value 저장, key 또는 value로 데이터 정렬 가능

```
from collections import OrderedDict

def sort_by_key(t):
      return t[0] #t[0]-key, t[1]-value

d = dict()
d['b'] = 100
d['a'] = 700
d['d'] = 300
d['c'] = 500
d['e'] = 200

for key,value in OrderedDict(sorted(d.items(), key=sort_by_key)).items():
      print(key,value)
```

```
a 700
b 100
c 500
d 300
e 200
```

collections 모듈

- collections – 파이썬 내장 모듈(build-in data structure)

  Counter : 데이터 값의 개수를 딕셔너리 형태로 반환

```
from collections import Counter

fruits = ['apple', 'banana', 'cherry', 'apple', 'cherry', 'apple', 'banana']

fruit_count = Counter(fruits)

print(fruit_count)
```

```
Counter({'apple': 3, 'banana': 2, 'cherry': 2})
```