Professional Coding Specialist

COS Pro 파이썬 1 급

3 강. 문법 정리 3

1. 2 차원 리스트와 튜플, 딕셔너리, 세트
2. 정렬 함수/메소드, enumerate, zip 함수

과정 소개

COS Pro 1 급 파이썬 시험 대비를 위한 문법 정리 세 번째 시간으로 리스트와 더불어 파이썬에서 제공하는 독특한 자료 구조인 튜플, 딕셔너리, 세트의 특징과 활용법에 대해서 정리한다. 그리고 이러한 자료구조에 정렬 함수나 메소드를 적용한 데이터 정렬법을 익히고, 항목 값과 순서 값을 동시에 가져오게 하는 enumerate( ) 함수와 두 개의 리스트에서 항목 값을 한꺼번에 가져오는 zip( ) 함수의 사용법을 학습하여 COS Pro 1 급 파이썬 문제 풀이에 활용할 수 있도록 한다.

학습 목차

1. 2 차원 리스트와 튜플, 딕셔너리, 세트
2. 정렬 함수/메소드, enumerate, zip 함수

학습 목표

1. 2 차원 리스트를 생성하고 그 리스트의 항목을 가져와서 활용할 수 있다.
2. 튜플, 딕셔너리, 세트를 생성하고 각각의 특징에 맞게 해당 자료구조를 활용한 프로그래밍을 할 수 있다.
3. 리스트 메소드 sort( )와 파이썬 내장함수 sorted( )를 사용하여 리스트의 항목 값을 오름차순/내림차순으로 정렬하고 메소드 sort( )와 내장함수 sorted( )의 차이점을 이해할 수 있다.
4. 리스트 메소드 reverse( ) 와 파이썬 내장함수 reversed( ) 를 사용하여 리스트의 항목들을 역순으로 가져오고, 메소드 reverse( )와 내장함수 reversed( )의 차이점을 설명할 수 있다.
5. enumerate( ) 함수를 이용하여 리스트의 인덱스와 항목 값을 한 번에 가져와서 활용할 수 있다.
6. zip( ) 함수를 이용하여 두 개의 리스트에서 항목을 동시에 가져와 비교할 수 있다.

1 2 차원 리스트와 튜플, 딕셔너리, 세트

1. 2 차원 리스트

1) 특징

- 행과 열이 있는 테이블 구조.
- 리스트의 항목으로 리스트가 있는 구조.
- 리스트 안에 리스트 뿐만 아니라 튜플도 가능.

예) 한 학교의 학년 별, 반 별 인원을 관리하는 경우

구분	1 반	2 반	3 반	4 반
1 학년	27	28	27	30
2 학년	30	29	27	29
3 학년	27	28	29	30

➔ 각 항목 값을 나타내는 인덱스

[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]

➔ 파이썬 코드로 구현

```
g = [[27, 28, 27, 30],
     [30, 29, 27, 29],
     [27, 28, 29, 30]]

print("2학년 각 반의 인원 수",g[1])
print("2학년 3반의 인원 수:",g[1][2])
```

```
2학년 각 반의 인원 수 [30, 29, 27, 29]
2학년 3반의 인원 수: 27
```

2) 2 차원 리스트 출력하기

① 중복 for 문 사용

```
g = [[27, 28, 27, 30],
     [30, 29, 27, 29],
     [27, 28, 29, 30]]

for i in range(3):
    for j in range(4):
        print(i+1,"학년", j+1,"반",g[i][j],"명")
```

```
1 학년 1 반 27 명
1 학년 2 반 28 명
1 학년 3 반 27 명
1 학년 4 반 30 명
2 학년 1 반 30 명
2 학년 2 반 29 명
2 학년 3 반 27 명
2 학년 4 반 29 명
3 학년 1 반 27 명
3 학년 2 반 28 명
3 학년 3 반 29 명
3 학년 4 반 30 명
```

② 행 전체의 항목들을 변수 하나에 받아서 출력.

```
g = [[27, 28, 27, 30],
     [30, 29, 27, 29],
     [27, 28, 29, 30]]

for grade in g:
    print(grade)
```

```
[27, 28, 27, 30]
[30, 29, 27, 29]
[27, 28, 29, 30]
```

③ 행에 있는 항목들을 하나씩 변수로 받아와서 출력.

```
g = [[27, 28, 27, 30],
     [30, 29, 27, 29],
     [27, 28, 29, 30]]

for grade in g:
    for c in grade:
        print(c, end=' ')
    print()
```

```
27 28 27 30
30 29 27 29
27 28 29 30
```

```
g = [[27, 28, 27, 30],
     [30, 29, 27, 29],
     [27, 28, 29, 30]]

for c1, c2, c3, c4 in g:
    print(c1, c2, c3, c4)
```

```
27 28 27 30
30 29 27 29
27 28 29 30
```

2. 튜플(tuple)

1) 특징

- 리스트와 유사한 구조로 리스트와 달리 항목 값을 변경할 수 없음.

튜플명 = (값 1, 값 2, …)

예시)

basket = ('apple', 'orange', 'pear')

basket = ('apple',)

basket = ( )

※ 튜플에 하나의 항목 값만 지정하여 생성하는 경우, 괄호를 사용하는 다른 수식과 혼동하지 않도록 튜플 항목 값 뒤에 쉼표(,) 를 적어주는 규칙이 있음.

2) 튜플 활용 예

```
t1=tuple()
t2=()

t1=('apple','orange')

print(t1)
print(t1[0])
```

- 빈 튜플 생성
- 튜플 t1 에 값을 할당
- 튜플 t1 전체를 출력.

- 튜플 t1 의 0 번째 인덱스의 값을 출력.

```
('apple', 'orange')
apple
```

3) 튜플의 항목을 새로운 값으로 변경하면 오류 발생

```
t1=tuple()
t1=('apple','orange')
t1[0] ='mango'
```

- t1 의 0 번째 항목에 새로운 값을 할당하는 코드를 실행하면 오류 발생.
-

<오류 메시지>

TypeError : "tuple" object does not support item assignment

4) 튜플 패킹과 언패킹

tuple packing	묶기 – 소괄호나 *로 값을 묶거나 변수 하나에 두 개 이상의 값을 할당.
tuple unpacking	풀기 - 튜플의 항목을 각각의 변수에 할당하여 하나씩 가져오기.

① packing

```
p = (10, 20)
print(p)

p2 = 30, 40
print(p2)
```

- 소괄호로 묶은 값으로 튜플 생성
- 하나의 변수에 두 개의 값을 할당하여 튜플 생성.

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

② unpacking

```
p3 = (10, 20)
u1, u2 = p3
print(u1, u2)
```

- 2 개의 항목을 갖는 튜플 생성
- 두 개의 튜플 항목을 각각의 변수에 할당.

```
10 20
```

③ unpacking 과 packing

```
p4 = (10, 20, 30, 40, 50)
u1, u2, *u3 = p4
print("u1:", u1, "u2:", u2, "u3:", u3)
print(type(u1), type(u3))
```

- 5 개의 항목을 갖는 튜플 생성
- p4 의 첫 번째 항목은 u1, 두 번째 항목은 u2, 나머지 항목들은 u3 에 할당.
- u1, u2 는 정수형 변수로 u3 은 리스트형 변수로 값을 저장.

```
u1: 10 u2: 20 u3: [30, 40, 50]
<class 'int'> <class 'list'>
```

5) 함수에서의 패킹과 언패킹

① 함수에서의 packing

```
def func_pack():
    return 10, 20

n1=func_pack()
print(n1)
```

- 함수의 return 값은 두 개이면서 그 값을 저장하는 변수가 하나인 경우 n1 에는 튜플 형태로 저장.

```
(10, 20)
```

② 함수에서의 unpacking

```
def func_unpack(u1, u2, *u3):
    print("u1:", u1, "u2:", u2, "u3:", u3)
    print(type(u1), type(u3))

func_unpack(10, 20, 30, 40, 50)
```

func_unpack( ) 함수에 5 개의 인수를 전달하며 호출.

- 첫 번째, 두 번째 인수는 u1, u2 에 저장되고, 나머지 값들은 u3 에 튜플 형태로 저장.

```
u1: 10 u2: 20 u3: (30, 40, 50)
<class 'int'> <class 'tuple'>
```

3. 딕셔너리(dictionary)

1) 특징

- 키(key) 와 값(value) 의 쌍으로 이루어진 구조.
- 딕셔너리의 키(key)는 중복되는 값을 가질 수 없고, 튜플과 마찬가지로 변경 불가.
- 자료의 순서가 없어서 인덱스를 가지고 있지 않음.

딕셔너리명 = { 키 1:값 1, 키 2:값 2, ... }

※ 딕셔너리의 키(key)로 리스트는 사용할 수 없음.

예) 반 별 인원을 관리하는 경우

키(Key)	1 반	2 반	3 반
값(Value)	27	28	27

➔ 파이썬 코드로 구현

```
m = {"1반":27, "2반":28, "3반":27}
print(m["2반"])
```

- 딕셔너리 m 을 생성한 후 키 값이 "2 반"인 항목값을 출력.

- 28

```
for i in m:
    print(i, ":", m[i])
```

- for 문을 이용해 m 의 키 값을 하나씩 받아와 키와 키에 해당하는 항목값을 출력.

```
1반 : 27
2반 : 28
3반 : 27
```

```
m["3반"] = 40

print(m)
```

- 키 값이 "3 반"인 항목값을 40 으로 변경한 후 딕셔너리 m 을 출력.

```
{'1반': 27, '2반': 28, '3반': 40}
```

2) 딕셔너리 관련 함수, 메소드

① 딕셔너리 생성 함수

dict( )	딕셔너리를 생성하여 리턴.

```
m2={"A":27, "B":28, "C":27}
print(m2)

m3=dict(A=2, B=28, C=27)
print(m3)

m4=dict(A반=2, B반=28, C반=27)
print(m4)
```

- 중괄호를 이용하여 키와 값을 대응시킨 딕셔너리 생성.
- dict( ) 함수의 매개변수에 키워드 인수 값을 할당하는 것과 같은 형태로 알파벳으로 시작하는 키 값에 값을 대응시키는 형태로 dict( ) 함수를 사용하여 딕셔너리를 생성.

<결과>

```
{'A': 27, 'B': 28, 'C': 27}
{'A': 2, 'B': 28, 'C': 27}
{'A반': 2, 'B반': 28, 'C반': 27}
```

② 딕셔너리 메소드

keys( )	딕셔너리의 키들을 리턴.
values( )	딕셔너리의 값들을 리턴.
items( )	딕셔너리의 키와 값을 튜플로 묶은 항목으로 리턴.

```
m = {"1반":27, "2반":28, "3반":27}

print(m.keys())
print(m.values())
print(m.items())
```

- 딕셔너리 메소드 keys( ), values( ), items( )를 이용하여 값을 출력.

<결과>

```
dict_keys(['1반', '2반', '3반'])
dict_values([27, 28, 27])
dict_items([('1반', 27), ('2반', 28), ('3반', 27)])
```

```
m = {"1반":27, "2반":28, "3반":27}

chg_list1 = list(m.values())
print(chg_list1)

chg_list2=list(m.keys())
print(chg_list2)

chg_list3=list(m.items())
print(chg_list3)
```

- 딕셔너리의 값, 키, 키와 값의 쌍의 목록들을 출력할 때 리스트 형태로 가져올 수 있도록 리스트 타입으로 형 변환.

<결과>

```
[27, 28, 27]
['1반', '2반', '3반']
[('1반', 27), ('2반', 28), ('3반', 27)]
```

-

4. 세트(Set)

1) 특징

- 중복된 항목 값을 갖지 않고 각 항목의 순서가 존재하지 않음.

세트명 = { 항목 1, 항목 2, ... }

2) 세트의 생성

```
s1 = set("acaade")
print(s1)

s2={"a","e","a","d","d","c"}
print(s2)
```

- set( ) 함수의 인수로 전달된 문자열에 있는 문자들 중 중복된 문자는 뺀 나머지를 항목으로 갖는 세트를 생성.
- 중괄호를 이용하여 항목을 할당한 것 중에 중복된 항목은 제외한 것들로 세트를 생성.

<결과>

```
{'c', 'a', 'd', 'e'}
{'c', 'a', 'd', 'e'}
```

2 정렬 함수/메소드, enumerate, zip 함수

1. 정렬 함수와 메소드

1) 정렬 함수와 메소드 기능 정리

문법	설명 (a 가 리스트)
a.sort( )	항목들을 오름차순으로 정렬하고 리스트 항목들의 순서를 변경. 메소드
a.sort(reverse=True)	항목들을 내림차순으로 정렬하고 리스트 항목들의 순서를 변경. 메소드
a.sort(key=str.lower)	항목값에 알파벳이 있는 경우 소문자로 변경한 기준으로 정렬. 메소드(대문자로 변경해서 정렬하는 경우는 str.upper 로 지정)
a.reverse( )	리스트 항목들의 순서를 역순으로 재배치. 메소드
sorted(a)	a 의 항목들을 오름차순으로 정렬한 것으로 리턴. 리스트 a 자체의 값은 변경되지 않음. 함수.
sorted(a, reverse=True)	a 의 항목들을 내림차순으로 정렬한 것으로 리턴. 리스트 a 자체의 값은 변경되지 않음. 함수.
reversed(a)	a 의 항목들을 역순으로 재배치한 것을 리턴. 리스트 a 자체의 값은 변경되지 않음. 함수.

2) 정렬 함수와 메소드 사용 예 : a = [9, 3, 1, 15] 인 경우

① sort( ) 메소드를 활용한 오름차순 정렬

```
a=[9,3,1,15]

a.sort( )
print("===a.sort()후 a")
print(a, "₩n")
```

- 리스트 a 를 오름차순으로 정렬한 값으로 재배치.

<결과>

```
===a.sort()후 a
[1, 3, 9, 15]
```

② sort( ) 메소드를 활용한 내림차순 정렬

```
a=[9,3,1,15]

a.sort(reverse=True)
print("===a.sort(reverse=True)후 a")
print(a, "₩n")
```

- 리스트 a 를 내림차순으로 정렬한 값으로 재배치.

<결과>

```
===a.sort(reverse=True)후 a
[15, 9, 3, 1]
```

③ reverse( ) 메소드를 활용해서 역순으로 항목 재배치

```
a=[9,3,1,15]

a.reverse()
print("===a.reverset()후 a")
print(a, "₩n")
```

- 리스트 a 를 역순으로 항목들을 재배치.

<결과>

```
===a.reverset()후 a
[15, 1, 3, 9]
```

④ sorted( ) 함수를 활용해서 오름차순으로 정렬

```
a=[9,3,1,15]

print("===sorted(a)")
print(sorted(a),"₩n")
print("===sorted(a)후 a")
print(a,"₩n")
```

- 리스트 a 를 오름차순으로 정렬한 값을 출력.
- 리스트 a 출력. → sorted( ) 함수를 적용한 후에도 리스트 a 에는 변화 없음.

<결과>

```
===sorted(a)
[1, 3, 9, 15]

===sorted(a)후 a
[9, 3, 1, 15]
```

⑤ sorted( ) 함수를 활용해서 내림차순으로 정렬

```
a=[9,3,1,15]

print("===sorted(a)")
print(sorted(a, reverse=True),"₩n")
print("===sorted(a, reverse=True)후 a")
print(a,"₩n")
```

- 리스트 a 를 내림차순으로 정렬한 값을 출력.
- 리스트 a 출력. → sorted( ) 함수를 적용한 후에도 리스트 a 에는 변화 없음.

<결과>

```
===sorted(a)
[1, 3, 9, 15]

===sorted(a)후 a
[9, 3, 1, 15]
```

⑥ reversed( ) 함수를 활용해서 리스트의 항목들을 역순으로 출력

```
a=[9,3,1,15]

print("===reversed(a)")
print(reversed(a))
print(list(reversed(a)),"₩n")

print("---reversed(a) 후a")
print(a,"₩n")
```

- 리스트 a 를 역순으로 재배치한 객체 출력.
- 리스트 a 를 역순으로 재배치한 객체를 리스트로 형 변환 하여 출력.
- 리스트 a 출력. → reversed( ) 함수를 적용한 후에도 리스트 a 에는 변화 없음.

<결과>

```
===reversed(a)
<list_reverseiterator object at 0x00000201971DD3D0>
[15, 1, 3, 9]

---reversed(a) 후a
[9, 3, 1, 15]
```

3) Timsort 알고리즘 : 파이썬 sort( ) 에 사용하는 알고리즘

- list.sort( ) 와 sorted( ) 에 사용되는 Timsort 알고리즘은 key function 을 사용할 때 더 빠른 속도와 더 적은 메모리를 사용해서 정렬을 수행.
- 병합 정렬과 삽입 정렬 방식을 기초로 만들어진 알고리즘으로 파이썬 버전 2.3 부터 파이썬 표준 정렬 알고리즘으로 사용. (https://en.wikipedia.org/ 참고)

2. enumerate 함수

1) 기능

- 인덱스와 그 인덱스에 대응하는 항목 값을 튜플 형태로 동시에 return.
- 리스트, 튜플 같이 항목이 여러 개인 데이터에 대해서 사용.

2) 사용 예

```
x = ['a','b','c']

e=list(enumerate(x))
print(e)
```

리스트 x 의 인덱스와 항목 값을 튜플로 묶어 온 것을 리스트로 변환.

```
[(0, 'a'), (1, 'b'), (2, 'c')]
```

```
x = ['a','b','c']

for i, v in enumerate(x):
    print(i, v)
```

리스트 x 의 인덱스와 항목 값 for 문을 이용해 I, v 로 받아서 출력.

```
0 a
1 b
2 c
```

※ 참조 : Python Document
(https://docs.python.org/3/library/functions.html?highlight=enumerate#enumerate)

3. zip 함수

1) 기능

- 두 개의 리스트 항목을 한꺼번에 가져오기 위해 사용.
- 인덱스가 같은 항목 값을 튜플 형태로 묶어서 return 하기 때문에 두 리스트에서 같은 인덱스의 항목 값을 쉽게 비교할 수 있음.

2) 사용 예

```
x1 = [1,2,3]
x2 = [4,5,6]

z = list(zip(x1, x2))
print(z)
```

- 같은 인덱스를 갖는 x1, x2 의 항목끼리 묶어 튜플로 만들 것을 리스트로 변환.

```
[(1, 4), (2, 5), (3, 6)]
```

```
x1 = [1,2,3]
x2 = [4,5,6]

for a, b in zip(x1, x2):
    print(a, b)
```

- x1, x2 에서 같은 인덱스를 갖는 항목을 for 문을 이용하여 a, b 로 받아와서 출력.

```
1 4
2 5
3 6
```

※ 참조 : Python Document
(https://docs.python.org/3/library/functions.html?highlight=enumerate#zip)