리스트 자료형의 활용법

1 리스트의 기본 사용법

- 대괄호로 정의
- 내장 함수 사용

```
l = []
print(type(l))
```

```
<class 'list'>
```

```
l = list()
print(type(l))
```

```
<class 'list'>
```

리스트 자료형의 활용법

1 리스트의 기본 사용법

리스트 안의 리스트

- 다차원 배열로 사용 가능

```
l = [[1,2,3],[4,5,6],[7,8,9]]
print(l)
```

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

```
print(l[0])
print(l[0][1])
```

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

리스트 자료형의 활용법

2 리스트의 특징

insert 함수 해당 인덱스에 값 추가

```
l = [9,7,4,7,2,6,8,9]
print(l)
```

```
[9, 7, 4, 7, 2, 6, 8, 9]
```

인덱스

```
l.insert(0,1)
print(l)
```

```
[1, 9, 7, 4, 7, 2, 6, 8, 9]
```

값

```
l.insert(4,'안녕하세요')
print(l)
```

```
[1, 9, 7, 4, '안녕하세요', 7, 2, 6, 8, 9]
```

리스트 자료형의 활용법

2 리스트의 특징

pop 함수 해당 인덱스의 값 삭제

```
l = [9,7,4,7,2,6,8,9]
print(l)
```

```
[9, 7, 4, 7, 2, 6, 8, 9]
```

```
l.pop()
print(l)
```

```
[9, 7, 4, 7, 2, 6, 8]
```

```
l.pop(0)
print(l)
```

```
[7, 4, 7, 2, 6, 8]
```

리스트 자료형의 활용법

2 리스트의 특징

reverse 함수 값을 거꾸로 정렬

```
l = [1,2,3,4,5,6,7,8,9]
print(l)
```

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

```
l.reverse()
print(l)
```

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

리스트 자료형의 활용법

2 리스트의 특징

count 함수	해당 값의 개수

```
l.count(1)
```

```
1
```

```
l.count(10)
```

```
0
```

리스트 자료형의 활용법

2 리스트의 특징

append 함수	해당 값을 요소로 추가
extend 함수	해당 리스트를 뒤에 가져다 붙임(확장)
sort 함수	오름차순으로 값 정렬 (정렬 방식 변경 가능)
index 함수	리스트에 해당 값이 있으면 해당 값의 인덱스 반환
copy 함수	리스트를 복사

리스트 자료형의 활용법

3 리스트의 활용

① 리스트 내포

리스트를 만드는 반복문을 짧고 간결하게 만들어 줌

[식 for 변수 in 리스트(튜플, 문자열, 사전 등)]

```
l = []
for i in range(5):
    #파이썬의 반복문
    l.append(i)
print(l)
```

```
[0, 1, 2, 3, 4]
```

0~4 숫자를 하나씩 꺼내
리스트로 생성

```
print([i for i in range(5)])
```

```
[0, 1, 2, 3, 4]
```

리스트 자료형의 활용법

3 리스트의 활용

② 리스트를 큐(Queue)로 활용

큐(Queue)

먼저 집어 넣은 데이터가 먼저 나오는 FIFO(First in First Out) 구조로 저장하는 형식

1 리스트에 데이터 추가하기

- 데이터는 뒤에서 추가되어야 하기 때문에 리스트의 append 함수를 사용

```
l = [1,2,3,4,5]
```

```
l.append(6)
print(l)
```

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

리스트 자료형의 활용법

3 리스트의 활용

② 리스트를 큐(Queue)로 활용

2 리스트에서 데이터 하나씩 빼기

- 데이터는 앞에서부터 하나씩 빠져야 하기 때문에 리스트의 pop 함수를 사용

```
l = [1,2,3,4,5]
```

```
l.pop(0)
print(l)
```

```
[2, 3, 4, 5, 6]
```

리스트 자료형의 활용법

3 리스트의 활용

③ 리스트를 스택(Stack)으로 활용

스택(Stack)

나중에 집어 넣은 데이터가 먼저 나오는 LIFO(Last in First Out) 구조로 저장하는 형식

1 리스트에 데이터 추가하기

- 데이터는 뒤에서 추가되어야 하기 때문에 리스트의 append 함수를 사용

```
l = [1,2,3,4,5]
```

```
l.append(6)
print(l)
```

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

리스트 자료형의 활용법

3 리스트의 활용

③ 리스트를 스택(Stack)으로 활용

2 리스트에서 데이터 하나씩 빼기

- 데이터는 뒤에서부터 하나씩 빠져야 하기 때문에 리스트의 pop 함수를 사용

```
l = [1,2,3,4,5]
```

```
l.pop()
print(l)
```

```
[1, 2, 3, 4, 5]
```

튜플 자료형의 활용법

1 튜플의 기본 사용법

- 소괄호로 정의
- 내장 함수 사용

```
t = ()
print(type(t))
```

```
<class 'tuple'>
```

```
t = tuple()
print(type(t))
```

```
<class 'tuple'>
```

튜플 자료형의 활용법

1 튜플의 기본 사용법

원소가 하나인 튜플 정의

- 콤마 사용

```
t = (1)
print(type(t))
```

```
<class 'int'>
```

```
t = (1,)
print(type(t))
```

```
<class 'tuple'>
```

```
t = 1,
print(type(t))
```

```
<class 'tuple'>
```

튜플 자료형의 활용법

2 튜플의 특징

count 함수	해당 값의 개수

```
t = (1,2,2,2,3,3,4,5,6,7,8,9)
```

순서	0	1	2	3	4	5	6	7	8	9	10	11
값	1	2	2	2	3	3	4	5	6	7	8	9

```
print(t.count(2))
```

```
3
```

```
print(t.count(3))
```

```
2
```

```
print(t.count(0))
```

```
0
```

튜플 자료형의 활용법

2 튜플의 특징

index 함수

해당 값의 순서, 값이 여러 개일 땐 첫 번째

```
t = (1,2,2,2,3,3,4,5,6,7,8,9)
```

순서	0	1	2	3	4	5	6	7	8	9	10	11
값	1	2	2	2	3	3	4	5	6	7	8	9

```
print(t.index(5))
```

```
7
```

```
print(t.index(2))
```

```
1
```

```
print(t.index(9))
```

```
11
```

튜플 자료형의 활용법

3 튜플의 활용

값의 할당 및 리턴

```
t = 1,2,3
print(type(t))
print(t)
```

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

```
x,y,z = 1,2,3
print(x)
print(y)
print(z)
```

```
1
2
3
```

```
def test():
    #파이썬의 함수
    return (1,2)

a,b = test()
print(a)
print(b)
```

```
1
2
```

사전 자료형의 활용법

1 사전의 기본 사용법

- 중괄호로 정의
- 내장 함수 사용

```
d = {}
print(type(d))
```

```
<class 'dict'>
```

```
d = dict()
print(type(d))
```

```
<class 'dict'>
```

사전 자료형의 활용법

1 사전의 기본 사용법

사전 사용 시 주의사항
➡ 사전에서 키는 고유한 값이므로 중복되는 키에 값을 입력하면 기존 값이 사라짐

```
d[1] = 1
print(d)
```

```
{1: 1}
```

↓

```
d[1] = 2
print(d)
```

```
{1: 2}
```

사전 자료형의 활용법

2 사전의 특징

update 함수	두 사전의 병합

- a.update(b) : a사전에 b사전을 새롭게 업데이트하는 함수로 a값만 변경됨

```
dict1 = {'a': 1, 'b': 2, 'c': 3}
dict2 = {'c': 2, 'd': 4, 'e': 5}
```

```
dict1.update(dict2)
print(dict1)
```

```
{'a': 1, 'b': 2, 'c': 2, 'd': 4, 'e': 5}
```

사전 자료형의 활용법

2 사전의 특징

pop 함수 해당 키의 항목 삭제

```
d = {'a': 1, 'b': 2, 'c': 3}
print(d)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

```
d.pop('a')
print(d)
```

```
{'b': 2, 'c': 3}
```

clear 함수 전체 삭제

```
d.clear()
print(d)
```

```
{}
```

사전 자료형의 활용법

2 사전의 특징

copy 함수 사전의 복사

- 새로운 객체를 만들어서 복사

```
d = {'a' : 1, 'b' :2, 'c' : 3}
a = d.copy()
b = d
print(d)
print(a)
print(b)
```

```
{'a': 1, 'b': 2, 'c': 3}
{'a': 1, 'b': 2, 'c': 3}
{'a': 1, 'b': 2, 'c': 3}
```

```
a['a'] = 2
b['b'] = 3
print(d)
print(a)
print(b)
```

```
{'a': 1, 'b': 3, 'c': 3}
{'a': 2, 'b': 2, 'c': 3}
{'a': 1, 'b': 3, 'c': 3}
```

사전 자료형의 활용법

3 사전의 활용

① 반복문에서의 사전

```
d = {'a': 1, 'b': 2, 'c': 3}
```

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

```
a
b
c
```

```
for i in d.values():
    print(i)
```

```
1
2
3
```

```
for i in d.items():
    print(i)
```

```
('a', 1)
('b', 2)
('c', 3)
```

사전 자료형의 활용법

3 사전의 활용

② 사전에서 값의 유무 알아보기

```
dic = {'a' : 1, 'b' : 2, 'c' : 3}
print(dic)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

```
'a' in dic
```

```
True
```

```
'd' in d
```

```
False
```

↓

해당 키가 있으면 True 없으면 False

```
2 in dic.values()
```

```
True
```

```
5 in dic.values()
```

```
False
```

↓

해당 값이 있으면 True 없으면 False

Run! 프로그래밍

Mission 1 | 리스트 함수 활용

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

l.sort(reverse=True)

print(l)
```

Mission 2 | 두 사전을 하나의 사전으로 병합

```
dic1 = {'a' : 1, 'b' : 2, 'c' : 3}
dic2 = {'b' : 5, 'c' : 1, 'd' : 7}
```

정답

```
dic2.update(dic1)
print(dic2)
```