Chapter 6

자료형(리스트, 튜플, 딕셔너리)

리스트

1. 리스트를 이용하면 여러 개 값을 저장
2. 인덱스는 0부터 시작
3. 리스트는 [ ]를 이용해 만듬
4. [ ]안에 [ ]를 넣으면 차원이 증가
   - 변수명 = […] : 1차원 리스트
   - 변수명 = [[…], …] : 2차원 리스트
   - 변수명 = [[[…], …], …] : 3차원 리스트

```
1  fruits = ["banana", "apple", "orange", "grape"]
2  print(fruits)
```

```
['banana', 'apple', 'orange', 'grape']
```

```
1  numbers = [1,2,3,4,5]
2  print(numbers)
```

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

다차원 리스트

```
1  numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
2  print(numbers_2d)
```

```
[[1, 2, 3, 4, 5], [10, 20, 30, 40, 50], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
```

행 \ 열	0	1	2	3	4
0	1	2	3	4	5
1	10	20	30	40	50
2	1	3	5	7	9
3	2	4	6	8	10

```
1  numbers_2d_v = [[1,2,3], [10,20,30,40], [1,3,5,7,9], [2,4,6,8,10,12]]
2  print(numbers_2d_v)
```

```
[[1, 2, 3], [10, 20, 30, 40], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10, 12]]
```

행 \ 열	0	1	2	3	4	5
0	1	2	3			
1	10	20	30	40		
2	1	3	5	7	9	
3	2	4	6	8	10	12

리스트 다루기

방법	설명
listData = [ ]	리스트를 만들어 줌
len(listData)	리스트의 항목의 수를 반환
mix(listData), max(listData)	리스트에서 가장 작은(min) 항목과 가장 큰(max) 항목을 반환
listData[start:stop]	리스트의 start 위치부터 stop 위치까지 부분 데이터를 추출(stop 위치의 항목은 포함 안 됨)
listData.append(value)	list에 value를 추가
listData.clear()	list의 모든 항목을 삭제
listData.count(value)	리스트에서 value의 개수를 반환
listData.extend(newList)	list에 newList를 추가
+	두 리스트를 연결함
listData.index(value, position=0)	position위치 이후에서 value의 값이 있는 인덱스를 반환
listData.insert(index, value)	list의 index위치에 value를 삽입
listData.remove(value)	리스트에서 해당 값을 삭제
del listData[index]	리스트에서 인덱스를 이용해 항목을 삭제
listData.pop()	리스트에서 가장 마지막 항목을 반환하고 삭제
listData.reverse()	리스트의 항목들의 순서를 반대로 함
listData.sort(reverse=False)	리스트의 항목들을 정렬. reverse 속성을 True로 하면 내림차순으로 정렬

리스트 함수 예시

1. + : 두 리스트를 연결
2. * : 리스트를 곱한 수만큼 반복

```
1  numbers = [1,2,3,4,5]
2  numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

```
1  new_numbers = numbers + numbers
2  new_numbers
```

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

```
1  3*numbers
```

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

리스트 함수 예시

1. append() : 단일 항목을 맨 뒤에 추가
   리스트를 append 하면 리스트가 항목으로 추가됨
2. extend() : 리스트를 항목별로 맨 뒤에 추가
3. insert() : 지정한 인덱스 위치에 삽입

```
1  numbers.append(10)
2  numbers
```

```
[1, 2, 3, 4, 5, 10]
```

```
1  numbers.append([20, 30, 40, 50])
2  numbers
```

```
[1, 2, 3, 4, 5, 10, [20, 30, 40, 50]]
```

```
1  numbers = [1,2,3,4,5]
2  numbers.extend([10,20,30,40,50])
3  numbers
```

```
[1, 2, 3, 4, 5, 10, 20, 30, 40, 50]
```

```
1  numbers.insert(5, 100)
2  numbers
```

```
[1, 2, 3, 4, 5, 100, 10, 20, 30, 40, 50]
```

리스트 함수 예시

1. count() : 리스트에서 데이터의 개수를 반환
2. index() : 해당 항목의 위치 반환

항목을 찾지 못하면 에러 발생

```
1  numbers = [1,3,5,7,9]
2  numbers.index(5)
```

```
2
```

```
1  numbers = [1,3,5,7,9,1,2,3,4,5]
2  numbers.index(5,3)
```

```
9
```

```
1  numbers = [1,3,5,7,9,1,2,3,4,5]
2  numbers.index(6,3)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback
<ipython-input-24-f30a67067ef4> in <module>()
      1 numbers = [1,3,5,7,9,1,2,3,4,5]
----> 2 numbers.index(6,3)

ValueError: 6 is not in list
```

리스트 함수 예시

1. pop() : 맨 뒤의 항목 반환 및 삭제
2. remove() : 해당 항목 삭제
3. clear() : 모든 항목 삭제
4. 공용 함수 del()

```
numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
numbers.pop()
```

```
10
```

```
numbers
```

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

```
numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
numbers.remove(3)
numbers
```

```
[1, 2, 4, 5, 6, 7, 8, 9, 10]
```

```
numbers_2d = [[1,2,3,4,5], [10,20,30,40,50], [1,3,5,7,9], [2,4,6,8,10]]
```

```
numbers_2d.remove([1,2,3,4,5])
numbers_2d
```

```
[[10, 20, 30, 40, 50], [1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
```

```
numbers = [1,2,3,4,5,6,7,8,9,10]
```

```
numbers.clear()
numbers
```

```
[]
```

리스트 함수 예시

1. sort() : 정렬(reverse=True 속성을 이용하면 내림차순 정렬) (원본데이터를 변경)
2. reverse() : 역순으로 나열(내림차순 정렬이 아님)

```
1  numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
1  numbers.sort()
2  numbers
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
1  numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

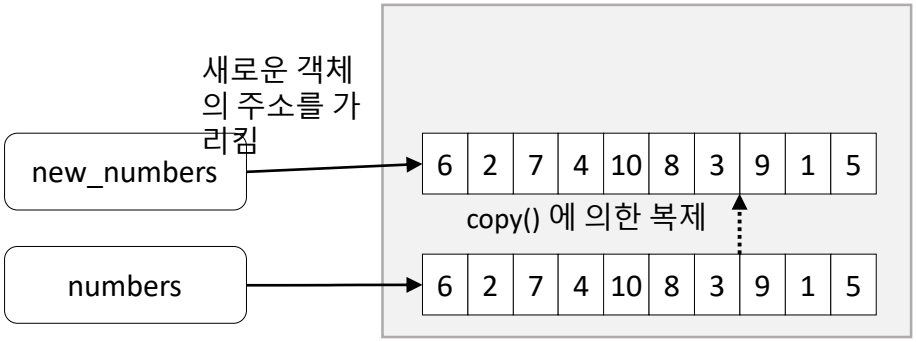
```
1  numbers.sort(reverse=True)
2  numbers
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

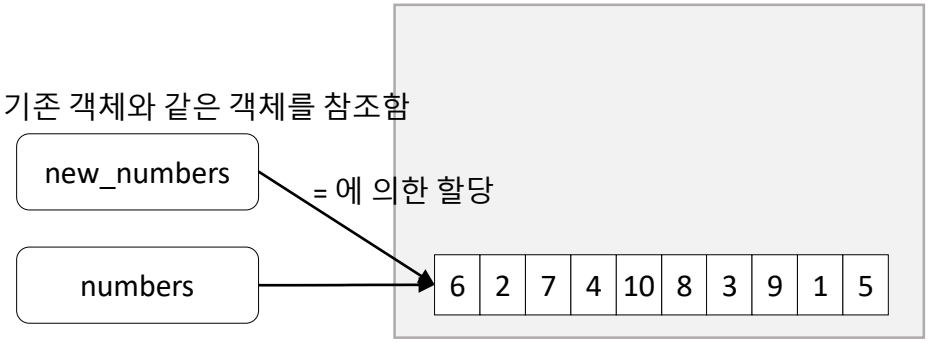
리스트 함수 예시

1. copy() : 복제된 새로운 객체를 생성
2. = : 주소를 복사해 같은 객체를 참조

```
1  numbers = [6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```



copy() 함수를 이용한 복사



할당연산자(=)를 이용한 복사

```
1  new_numbers = numbers.copy()
2  print(numbers)
3  print(new_numbers)
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

```
1  new_numbers = numbers
2  print(numbers)
3  print(new_numbers)
```

```
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
[6, 2, 7, 4, 10, 8, 3, 9, 1, 5]
```

튜플

1. 튜플(tuple)은 소괄호('('와 ')')를 이용해 만듬
2. 읽기 전용
3. 튜플은 속도가 빨라 수정이 필요 없는 배열 형태의 데이터 타입에 사용
4. 데이터를 수정할 수 없기 때문에 제공되는 함수가 많지 않음

방법	설명
tupleData = ( )	튜플을 만들어 줍니다.
len(tupleData)	튜플의 항목 수를 반환합니다.
min(tuple), max(tuple)	튜플에서 가장 작은 값(min)과 가장 큰 값(max)을 반환합니다.
tupleData.count(value)	튜플에서 value의 개수를 반환합니다.
tupleData.index(value, position)	position 위치 이후에서 value가 있는 인덱스를 반환합니다.

```
1  numbers1 = (1,2,3,4,5)
2  type(numbers1)
```

```
tuple
```

```
1  numbers2 = (1,)
2  type(numbers2)
```

```
tuple
```

```
1  numbers_2d = ((1,2,3), (4,5,6))
2  numbers_2d
```

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

튜플의 사용이유

- 튜플이 가능한 일은 리스트로도 모두 가능합니다.
- 리스트는 튜플에 비해 요소를 변경하는 편집도 가능합니다.

튜플의 사용이유!?

1. 비용의차이
리스트는 변경의 가능성을 항상 대비해야 되기때문에 더 많은 기능(메모리)를 소모합니다.
이에 비해, 튜플은 값의 바뀔 일이 없으므로 단순히 리스트해 비해 속도가 빠릅니다.

2. 데이터의 안정성
리스트 같은 경우는 실수로 데이터가 바뀔 위험성이 있는데, 튜플은 한번 정해지면 바꿀 수 없기 때문에 실수의 위험이 적습니다.

3. 리스트와 튜플은 상호 변경이 가능합니다.
값 변경 가능성 여부만 다를 뿐 구조는 유사하기 때문에 상호 변경이 가능합니다.

리스트 -> 튜플로 변경할 때 tuple()
튜플 -> 리스트로 변경할 때 list()

딕셔너리

1. 키(key)와 값(value)의 쌍으로 구성된 자료 구조
2. 딕셔너리를 만들기 위해서는 중괄호 {} 를 이용
3. 키는 중복이 없이 유일한 값이어야 함
4. 값은 중복이 가능하며 모든 타입이 가능
5. 인덱스를 이용한 데이터의 참조는 지원하지 음
6. 딕셔너리 키 목록에 없는 데이터를 사용하여 참조하면 에러가 발생

방법	설명
dictData = {“key”:“value”, ... }	딕셔너리를 만들어 줍니다.
len(dictData)	딕셔너리의 항목의 수를 반환합니다.
dictData.items()	딕셔너리의 각 항목들을 (key, value) 형식의 튜플들로 반환합니다.
dictData.keys()	딕셔너리의 키(key)들을 반환합니다.
dictData.values()	딕셔너리의 값(value)들을 반환합니다.

Chapter 6

수고하셨습니다