List, Tuple, Dictionary Type

Expression & Operation

송지수

List Type

Overview

✦많은 양의 데이터들을 하나로 묶어서 효율적으로 처리할 수 있는 방법

✦리스트 타입은 숫자나 문자열과 같은 다양한 자료형들을 주어진 순서에 따라 저장할 수 있는 데이터 타입

- Item : 정수, 실수, 문자열 등

✦리스트는 저장할 수 있는 모든 종류의 자료를 담을 수 있음

- 자료들을 섞어서 담을 수 있음

✦여러 개의 데이터를 모아서 하나로 다룰 수 있으므로 편리함

```
>>> family = ["엄마", "아빠", "형", "동생"]
>>> print(family)
['엄마', '아빠', '형', '동생']
```

List Type

Expression

✦리스트의 형태

- 각각의 데이터를 콤마로 구분하고 전체를 대괄호 '[]'로 묶어 시작과 끝을 표시함
- 리스트이름 = [데이터1, 데이터2, 데이터3, ...]

✦리스트의 초기화

- list1 = []
  - 아무 요소도 존재하지 않은 빈 리스트를 생성
- list2 = [10, 20, 30, 40, 20]
  - 일반적인 정수 값들을 가지는 리스트로 중복이 허용
- list3 = ["korea", "japan", "china"]
  - 문자열을 요소로 가진 리스트
- list4 = ["kim", 25, "lee", 26, "park", 30]
  - 서로 다른 자료형인 문자열과 정수를 요소로 가진 리스트
- list5 = [100, 200, ["morning", "evening"]]
  - 리스트 내에 또 다른 리스트를 요소로 가진 리스트

List Type

Expression

✦내장 함수 `list()`

- 이 함수는 어떤 순차 타입의 자료형을 리스트로 변환할 때 사용
- 이 함수를 사용하여 리스트 생성도 가능
- 형식 : `list(순차형자료)`

✦내장 함수 `range()`

- 해당 범위의 숫자들의 리스트를 반환하는 함수
- 형식 : `range(start, end, step)`
  - `start : 시작 값, end : 종료 값, step : 증가 값`
- 시작 값을 생략할 경우 0부터 생성하며, 종료 값은 리스트에 포함 안됨
- 사용 예 : `range(1, 5) → [1, 2, 3, 4]`

List Type

Expression

✦주의 사항

- range() 함수를 이용하여 규칙을 가진 연속된 정수들을 생성할 경우 단순히 연속된 정수들을 생성하기만 함
- 따라서 list()를 사용하여 range()의 반환 값을 리스트로 변환해야 함

```
>>> numbers1 = list()
>>> numbers2 = list("one")
>>> print(numbers1, numbers2)
[] ['o', 'n', 'e']
>>> numbers3 = list(range(10, 30, 2))
>>> print(numbers3)
[10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

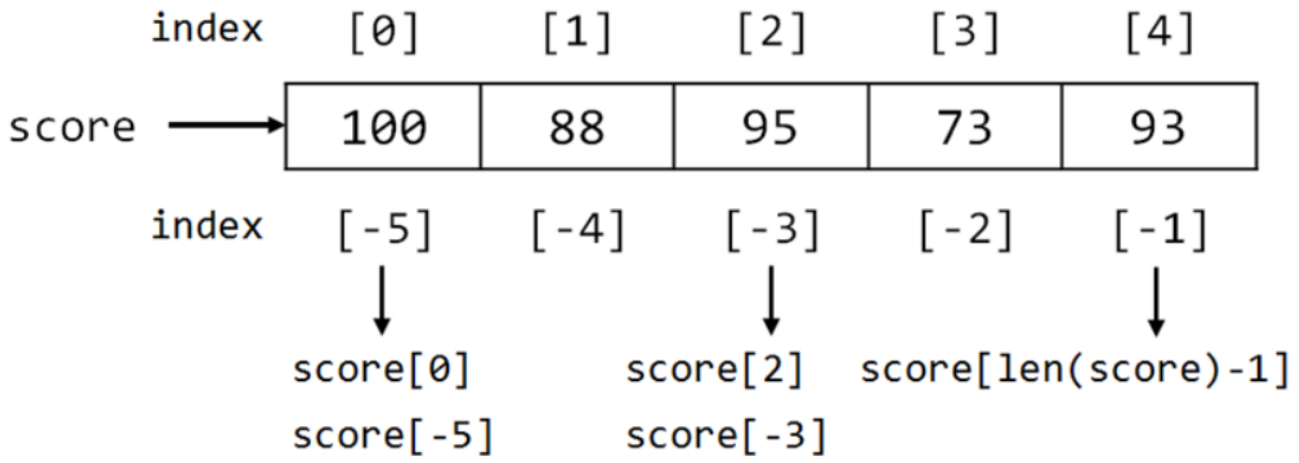
파이썬 v2에서 range()는 정수로 된 실제 리스트를 반환함
파이썬 v3에서 정수로 된 실제 리스트를 만들기 위해서는
range() 객체를 list() 함수를 사용해서 리스트로 변환해야 함

List Type

Indexing

✦리스트의 인덱싱 (Indexing)

- 리스트는 순차 자료형이므로 하나의 요소를 가져오기 인덱스를 이용하여 각 요소에 접근함
- 인덱스는 0부터 시작
- 인덱스가 음수일 경우 마지막 요소에서부터 거꾸로 접근함
- 만약 범위를 벗어난 요소를 접근할 경우 에러를 발생



Practice

Example 8-3

```
>>> score = [100, 88, 95, 73, 93]
>>> print(score[0], score[4], score[-1], score[-3])
100 93 93 95
>>> message = [10, 20, ["morning", "evening"]]
>>> print(message[2])
['morning', 'evening']

>>> number = [10, 20, 30]  리스트의 크기를 넘어서 접근한다면 오류를 발생
>>> number[3]
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    number[3]
IndexError: list index out of range
```

List Type

Indexing

✦리스트의 전체 요소를 가져오기 위한 방법

- for문이나 while문을 사용하여 리스트 요소를 차례로 가져옴

```
for score in [100, 98, 75, 86, 50]:
    print(score)
```

```
score = [100, 98, 75, 86, 50]
count = 0
while count < len(score):
    print(score[count])
    count += 1
```

※ len(list)
리스트내의 요소 개수를 구하는 함수

```
score = [100, 98, 75, 86, 50]
for count in range(score):
    print(score[count])
```

Practice

Example 8-4

✦서로 다른 리스트 요소를 반복문으로 출력하기

```
person = ["홍길동", 25, "박지성", 35, "박찬호", 32]

for count in range(len(person)):
    if count % 2 == 0:
        print("이름 : %s" %person[count], end=", ")
    else:
        print("나이 : %d" %int(person[count]))
```

```
이름 : 홍길동, 나이 : 25
이름 : 박지성, 나이 : 35
이름 : 박찬호, 나이 : 32
```

Practice

Example 8-5, 8-6

✦리스트로 주어진 점수들 중에 최대값 구하기

```
score = [93, 95, 88, 75, 98, 67]

max_score = score[0]
for item in score:
    if max_score < item:
        max_score = item

print("최대값 :", max_score)
```

```
max_score = score[0]
for count in range(1, len(score)):
    if max_score < score[count]:
        max_score = score[count]

print("최대값 :", max_score)
```

```
최대값 : 98
```

List Type

Slicing

✦리스트의 슬라이싱 (Slicing)

- 리스트도 슬라이싱을 통해 요소의 일부를 가져올 수 있음
- 리스트를 슬라이싱할 때 결과는 더 작은 리스트를 생성함
- 새로운 리스트는 새로운 객체로 인식하며, 원본은 그대로 유지
  - 기존의 리스트를 변경하는 것은 아니라 부분적으로 복사한 것임
- 형식 : `listobject[start : end : step]`
  - `listobject` : 리스트 객체 이름
  - `start` : 시작 인덱스, `end` : 마지막 인덱스, step : 리스트 요소를 가져오는 간격
  - 리스트의 부분 범위는 start에서 end의 바로 앞까지
- 인덱스 표시를 생략할 경우
  - `listobject[:]` → 전체의 리스트를 가져옴
  - `listobject[:3]` → 0(처음)부터 3번째 인덱스 요소 앞까지
  - `listobject[3:]` → 3번째부터 마지막 요소까지

Practice

Example 8-8

✦주어진 리스트의 수만큼 도형으로 출력하기

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

for item in frequency[:4]:
    for star in range(item):
        print("*", end="")
    print()

for item in frequency[4:6]:
    for star in range(item):
        print("#", end="")
    print()

for item in frequency[6:]:
    for star in range(item):
        print("*", end="")
    print()
```

```
*
**
***
****
#####
#####
****
***
**
*
```

List Type

Indexing & Slicing

✦리스트 요소의 변경

- 리스트는 문자열과는 다르게 데이터 요소의 변경이 가능함
- 슬라이싱으로 리스트를 확장하거나 축소할 수 있음

✦사용 예

- `numbers = [10, 20, 30, 40, 50]`
- `numbers[1] = 60 → [10, 60, 30, 40, 50]`
- `numbers[1] = [60, 70] → [10, [60, 70], 30, 40, 50]`
- `numbers[1:3] = [60, 70] → [10, 60, 70, 40, 50]`
- `numbers[1:3] = [60, 70, 80] → [10, 60, 70, 80, 40, 50]`
- `numbers[1:3] = [60] → [10, 60, 40, 50]`
- `numbers[1:3] = 60 → error`
- `numbers[1:3] = [] → [10, 40, 50]`
- `numbers = [] → []`

Practice

Example 8-10

✦문자열 요소를 수정하여 다시 문자열로 출력하기

```
message = "Hello Python"

message = list(message)
message[6] = "B"
message = "".join(message)
print(message)
```

```
Hello Bython
```

Practice

Example 8-11

✦0부터 15개의 피보나치수열 만들기

- 피보나치수열 조건 : F(0)=1, F(1)=1, F(n)=F(n-2)+F(n-1)

```
fibo = list(range(10))
fibo[0] = 1
fibo[1] = 1

for count in range(2, 10):
    fibo[count] = fibo[count-2] +
fibo[count-1]

print("피보나치 :", fibo)
```

```
피보나치 : [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

List Type

Operation

✦리스트의 결합

- 두 개의 리스트를 합칠 경우 '+' 연산자를 사용
- 하나의 리스트를 반복할 경우 '*' 연산자를 사용
- 연산 후 기존의 리스는 유지되며 새로운 리스트가 형성됨
  - `number1 = [10, 20, 30];  number2 = [40, 50, 60]`
  - `number1 + number2 → [10, 20, 30, 40, 50, 60]`
  - `number1 * 3 → [10, 20, 30, 10, 20, 30, 10, 20, 30]`

✦리스트의 분리

- 하나의 리스트를 여러 개의 리스트로 분리 (언팩킹의 개념을 사용)
  - `number1, *number2 = [10, 20, 30, 40, 50]`
    - `number1 = 10, number2 = [20, 30, 40, 50]`
  - `*number1, number2 = [10, 20, 30, 40, 50]`
    - `number1 = [10, 20, 30, 40], number2 = 5`

Practice

Example 8-13

```
>>> number1 = [10, 20, 30]
>>> number2 = [40, 50, 60]
>>> numbers = number1 + number2
>>> print(numbers)
[10, 20, 30, 40, 50, 60]
>>> numbers = number1 * 2
>>> print(numbers)
[10, 20, 30, 10, 20, 30]
>>> number1, *number2 = numbers
>>> print(number1, number2)
10 [20, 30, 10, 20, 30]
>>> *number1, number2 = numbers
>>> print(number1, number2)
[10, 20, 30, 10, 20] 30
```

List Type

Operation

✦리스트의 요소의 비교

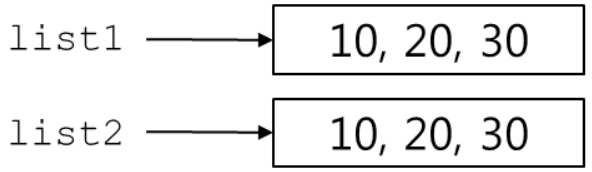
- 두 개의 리스트 요소를 비교하며, 동일한 자료형이어야 함
  - `[10, 20, 30] < [10, 20, 30, 40] → True`
  - `[10, 20, 40] < [10, 20, 30, 40] → False`
  - `[10, 20, 30] == [10, 20, 30] → True`

같은 값 인지 비교

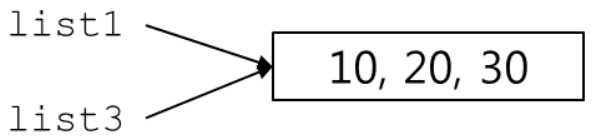
✦리스트 객체의 비교

- 두 개의 리스트가 같은 객체인지를 비교 (`is` 연산자)

```
>>> list1 = [10, 20, 30]
>>> list2 = [10, 20, 30]
```




```
>>> list1 = [10, 20, 30]
>>> list3 = list1
```



- `list1 is list2 → False`
- `list1 is list3 → True`

같은 객체 인지 비교

List Type

Operation

✦리스트의 얕은 복사 (Shallow Copy)

- 일반 대입 연산자를 사용한 복사 (Shallow Copy)
- 단지 객체 참조 변수를 복사함

```
>>> list1  = [10, 20, 30]
>>> list3 = list1
>>> list3[1] = 40
>>> print(list1)
[10, 40, 30]
```

원본도 같이 변경됨

✦리스트의 깊은 복사 (Deep Copy)

- 리스트의 완전한 복사로 리스트의 각 요소가 대응되어 복사
- `list()` 함수를 사용
- 복사 리스트 = `list`(원본 리스트)

Practice

Example 8-14

```
>>> [10, 20, 30] == [10, 20, 30]
True
>>> [10, 20, 30] < [10, 20, 30, 40]
True
>>> [10, 20, 40] < [10, 20, 30, 40]
False
>>> list1 = [1, 2, 3]
>>> list2 = [1, 2, 3]
>>> list1 is list2
False
>>> list3 = list1
>>> list1 is list3
True
>>> list4 = list(list1)
>>> list1 is list4
False
```

List Type

Function

✦리스트의 최대, 최소값 구하기

- `max()` 함수 : 리스트 내의 가장 큰 요소를 찾아 반환
- `min()` 함수 : 리스트 내의 가장 작은 요소를 찾아 반환

```
>>> score = [85, 93, 98, 75, 65, 90]
>>> print(max(score), min(score))
98 65
>>> names = ["홍길동", "박지성", "김연아"]
>>> print(max(names), min(names))
홍길동 김연아
```

List Type

Search

✦리스트 내의 요소 찾기

- in 연산자 : 리스트 내에 어떤 요소가 포함되어 있는지 조사
- not in 연산자 : 리스트 내에 어떤 요소가 포함되어 있지 않은지 조사
- 주로 if문을 사용하여 요소를 검사함
- True, False의 부울형 자료를 반환

✦index() 메서드

- 인수로 지정한 값이 리스트에서 처음 나타나는 인덱스를 반환
- 사용 예 : 리스트.index(찾을 요소, 찾을 위치)
  - 두 번째 인수에 찾을 위치에 대한 인덱스를 지정
- 찾는 값이 없을 경우 오류를 발생하므로 예외 처리가 필요함

✦count() 메서드

- 리스트 내의 요소들 중 특정 데이터가 몇 개나 있는지 확인
- 사용 예 : 리스트.count(찾을 요소)

Practice

Example 8-16

```
>>> score = [85, 93, 98, 75, 65, 90]
>>> 93 in score
True
>>> names = ["홍길동", "박지성", "김연아"]
>>> "홍길동" in names
True
>>> "홍" in names
False
```

```
>>> score = [85, 93, 98, 75, 65, 90]
>>> score.index(93)
1
>>> score.index(98, 2)
2
>>> score.index(92)
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    score.index(92)
ValueError: 92 is not in list
```

```
>>> score = [100, 98, 75, 100, 86, 100]
>>> score.count(100)
3
```

List Type

Function

✦enumerate() 함수

- 순차 타입의 자료를 입력 받아 요소와 인덱스를 동시에 얻음
- 순서 값과 요소로 구성된 enumerate 클래스의 객체를 반환함
- for문과 같은 반복문에서 현재 값과 인덱스가 필요한 경우 사용
- 사용 예 : enumerate(순차 타입의 자료)

```
score = [100, 98, 75, 86, 50]
for index in range(len(score)):
    print(index, score[index])
```

```
score = [100, 98, 75, 86, 50]
for index, item in enumerate(score):
    print(index, item)
```

List Type

List Comprehension

✦[수식 for 변수 in 리스트 if 조건]

- 리스트 안의 요소가 일정한 규칙을 가지는 수열이라면 일일이 나열할 필요 없이 List Comprehension 문법 적용

```
[n for n in range(1, 11)] #[1,2,3,4,5,6,7,8,9,10]
```

```
[n for n in range(1, 11) if n % 3 == 0] #[3,6,9]
```

Practice

Example 8-17

✦이름과 성별로 구성된 리스트에서 남녀 수와 이름을 출력하기

```
person = ["홍길동", "남", "박지성", "남", "김연아", "여", "박찬호", "남"]

print("남자 : %d명" %person.count("남"))
print("여자 : %d명" %person.count("여"))

for count, name in enumerate(person):
    if count % 2 == 0:
        print("%d.%s" %(count//2+1, name))
```

```
남자 : 3명
여자 : 1명
1.홍길동
2.박지성
3.김연아
4.박찬호
```

List Type

Function

✦all() 함수

- 순차 타입의 자료형에 대해 모든 요소가 참이면 True, 하나라도 거짓이면 False를 반환함
  - 요소가 정수라면 True, 0이나 빈 리스트이면 False를 뜻함
- 사용 예 : all(리스트)

✦any() 함수

- 순차 타입의 자료형에 대해 하나의 요소라도 참이면 True, 모두가 거짓이면 False를 반환함
- 사용 예 : any(리스트)

```
>>> result = [True, True, True]
>>> print(all(result), any(result))
True True
>>> result = [True, True, False]
>>> print(all(result), any(result))
False True
>>> score = [100, 90, 80, 0]
>>> print(all(score), any(score))
False True
```

List Method

Insert Item

✦리스트 요소의 삽입

- 리스트.append(데이터)
  - 리스트의 마지막 요소로 새로운 요소를 추가
  - 새로운 리스트를 생성하여 추가할 경우 먼저 공백 리스트를 생성함

```
>>> nation = []
>>> nation.append("korea")
>>> nation.append("japan")
>>> nation.append([10, 20])   리스트를 요소로 추가
>>> print(nation)
['korea', 'japan', [10, 20]]
```

- 리스트.insert(인덱스, 데이터)
  - 리스트에 삽입할 위치를 지정하여 새로운 요소를 추가
  - 삽입한 위치에 워래 존재하던 요소는 하나씩 뒤로 이동

```
>>> nation = ["korea", "japan"]
>>> nation.insert(1, "china")    1번 인덱스에 삽입
>>> nation.insert(1, "england")
>>> print(nation)
['korea', 'england', 'china', 'japan']
```

List Method

Insert Item

✦리스트 요소의 삽입

- 리스트.extend(확장할 리스트)
  - 리스트의 마지막에 다른 리스트의 요소를 새롭게 추가하여 확장
  - 새로운 리스트를 생성하지 않고 기존의 리스트가 변경됨

```
>>> nation = ["korea", "japan"]
>>> nation.extend(["china", "england"])
>>> print(nation)
['korea', 'japan', 'china', 'england']
>>> nation.append(["france", "america"])
>>> print(nation)
['korea', 'japan', 'china', 'england', ['france', 'america']]
```

Practice

Example 8-22

✦학생 5명의 점수를 입력 받아 리스트에 저장하고 평균 구하기

```
student = []
total = 0

for count in range(5):
    score = int(input("점수를 입력 :
"))
    student.append(score)
    total += score

average = total / len(student)
print("학생들의 점수 :", student)
print("평균 : %.1f" %average)
```

```
점수를 입력 : 95
점수를 입력 : 85
점수를 입력 : 98
점수를 입력 : 76
점수를 입력 : 88
학생들의 점수 : [95, 85, 98, 76, 88]
평균 : 88.4
```

Practice

Example 8-24

✦0에서부터 30개의 짝수를 리스트로 생성하여 추가하기

- 이 리스트에서 3의 배수 10개를 구해서 새로운 리스트를 만들기

```
number_list1 = []
number_list2 = []
item = 0

for count in range(30):
    number_list1.append(item)
    item += 2

for item in number_list1:
    if item % 3 == 0:
        number_list2.append(item)
    if len(number_list2) == 10:
        break

print("List1 =", number_list1)
print("List2 =", number_list2)
```

```
List1 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,
30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58]
List2 = [0, 6, 12, 18, 24, 30, 36, 42, 48, 54]
```

List Method

Delete Item

✦리스트 요소의 삭제

- del 리스트명[인덱스] or del 리스트명[슬라이싱]
  - 리스트 요소의 인덱스를 사용하여 리스트 항목을 삭제

```
>>> numbers = [10, 20, 30, 40, 50]
>>> del numbers[1]
>>> print(numbers)
[10, 30, 40, 50]
>>> del numbers[1:3]
>>> print(numbers)
[10, 50]
>>> del numbers
```

List Method

Delete Item

✦리스트 요소의 삭제

- 리스트.remove(삭제할 요소)
  - 인수에 삭제하고자 하는 요소를 직접 지정하여 리스트 항목을 삭제
  - 삭제된 후 자기 자신의 리스트는 변경됨
  - 만약 삭제할 요소가 실제 리스트에 존재하지 않을 경우 오류를 발생하므로 삭제 전에 삭제할 요소의 존재 유무에 대한 검사가 필요함

```
if 삭제할 요소 in 리스트:
    리스트.remove(삭제할 요소)
else:
    print("값이 존재하지 않습니다")
```

List Method

Delete Item

✦리스트 요소의 삭제

- 리스트.pop()
  - 메서드에 인수를 지정하지 않을 경우 리스트의 마지막 요소를 찾아서 삭제
  - 삭제 후에는 자기 자신의 리스트는 변경됨
  - 참고 : remove()의 경우 삭제 값을 반환하지 않음
- 리스트.pop(인덱스)
  - 리스트 내의 특정 요소를 삭제할 경우 메서드의 인수에 인덱스를 지정

```
>>> numbers = [10, 20, 30, 40, 50]
>>> del_item = numbers.pop()
>>> print(del_item, numbers)
50 [10, 20, 30, 40]
>>> del_item = numbers.pop()
>>> print(del_item, numbers)
40 [10, 20, 30]
>>> del_item = numbers.pop(1)
>>> print(del_item, numbers)
20 [10, 30]
```

List Method

Sort

✦정렬 (Sorting)

- 오름차순 : 데이터의 크기가 작은 쪽에서부터 큰 쪽으로 순서를 나열
- 내림차순 : 데이터의 크기가 큰 쪽에서 작은 쪽으로 순서를 나열

✦리스트 요소의 정렬

- sorted(순차타입의 자료) 내장 함수
  - 기본적으로 오름차순으로 데이터를 정렬하고 리스트의 복사본을 얻음
  - 원본 리스트는 변경되지 않음
- sorted(순차타입의 자료, reverse=True)
  - 내림차순으로 정렬할 경우 인수에 reverse키워드를 사용하여 True로 지정
- sorted() 함수에 문자열이 올 경우
  - 문자열의 각 문자를 요소로 하는 리스트를 생성하여 정렬함
  - sorted("python") → ['h', 'n', 'o', 'p', 't', 'y']
- 사용 예 : newlist = sorted(oldlist)

Practice

Example 8-30

```
>>> numbers = [1, 5, 7, 3, 6, 2, 8]
>>> print(sorted(numbers))
[1, 2, 3, 5, 6, 7, 8]
>>> print(sorted(numbers, reverse=True))
[8, 7, 6, 5, 3, 2, 1]
>>> message = "python"
>>> print(sorted(message))
['h', 'n', 'o', 'p', 't', 'y']
```

List Method

Sort

✦리스트 요소의 정렬

- 리스트.sort()
  - 리스트 객체가 가지고 있는 메서드로 인수가 생략될 경우 오름차순으로 정렬
- 리스트.sort(reverse=True)
  - 내림차순으로 정렬할 경우 인수에 reverse 키워드를 True로 지정

```
>>> numbers = [1, 5, 7, 3, 6, 2, 8]
>>> numbers.sort()
>>> print(numbers)
[1, 2, 3, 5, 6, 7, 8]
>>> numbers.sort(reverse=True)
>>> print(numbers)
[8, 7, 6, 5, 3, 2, 1]
```

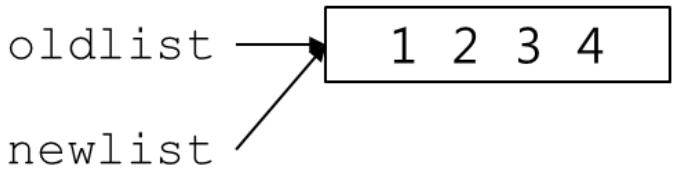
List Method

Sort

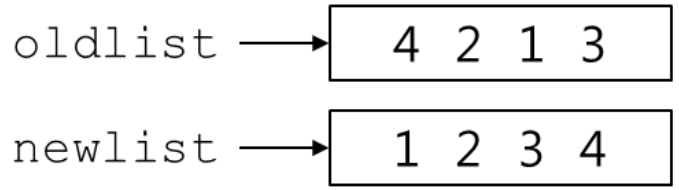
✦리스트 요소의 정렬

- sort() 메서드는 새로운 리스트를 만들지 않고 기존의 리스트를 정렬하여 원본을 수정하므로 반환 값이 발생하지 않음
- 따라서 정렬 전 리스트를 다시 사용하고자 한다면 미리 복사본을 준비하여야 함

```
>>> oldlist = [4, 2, 1, 3]
>>> newlist = oldlist
>>> newlist.sort()
>>> print(oldlist, newlist)
[1, 2, 3, 4] [1, 2, 3, 4]
```




```
>>> oldlist = [4, 2, 1, 3]
>>> newlist = oldlist[:]
>>> newlist.sort()
>>> print(oldlist, newlist)
[4, 2, 1, 3] [1, 2, 3, 4]
```



List Method

Sort

✦리스트 요소의 정렬

- 리스트.reverse()
  - 리스트의 요소들을 단순히 순서를 바꾸어 역순으로 정렬

```
>>> numbers = [1, 5, 7, 3, 6, 2, 8]
>>> numbers.reverse()
>>> print(numbers)
[8, 2, 6, 3, 7, 5, 1]
```

Practice

Example 8-33

✦5개의 국가를 입력하여 자동으로 리스트가 정렬되는 프로그램

```
nation = []
for count in range(5):
    nation.append(input("국가 입력 : "))
    nation.sort()

    for number in range(len(nation)):
        print("%d.%s" %(number+1, nation[number]), end=" ")
    print()
```

```
국가 입력 : korea
1.korea
국가 입력 : england
1.england 2.korea
국가 입력 : china
1.china 2.england 3.korea
국가 입력 : japan
1.china 2.england 3.japan 4.korea
국가 입력 : france
1.china 2.england 3.france 4.japan 5.korea
```

Nested List

Expression

✦중첩 리스트 (Nested List)

- 리스트에 순차 타입의 다른 객체를 요소로 가짐
- 리스트에 여러 개의 데이터를 가진 또 다른 리스트를 요소로 가짐
- 중첩 리스트에서 리스트의 요소는 내부 리스트 요소를 직접 저장하는 것이 아니라 객체 참조 값을 요소로 가짐

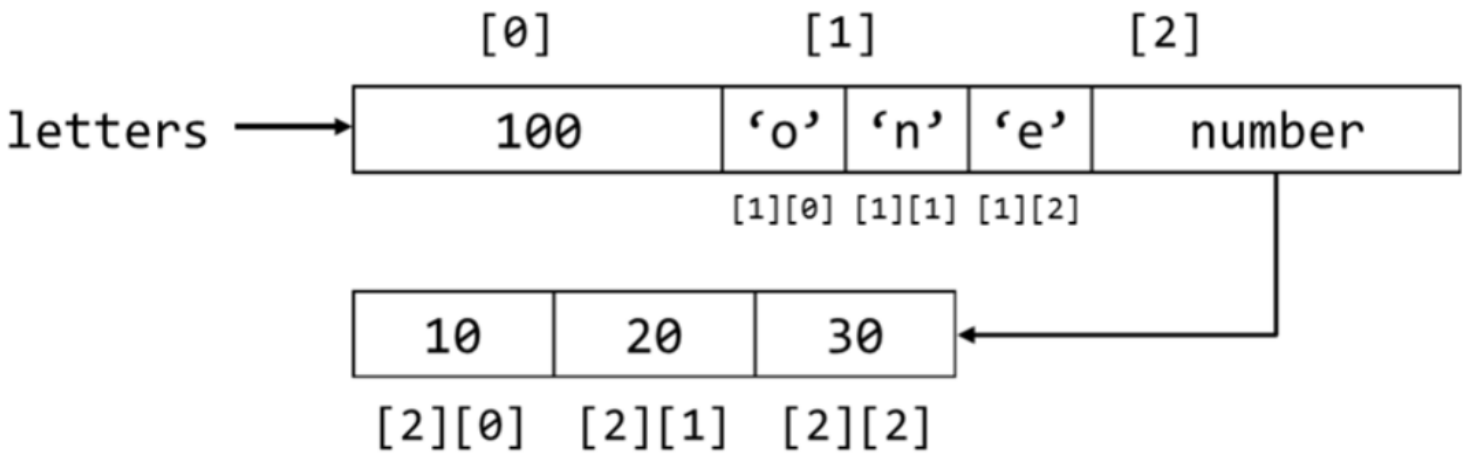
✦표현 방법

- 리스트에서 요소를 다른 리스트로 가짐
  - numbers = [10, 20, [1, 2, 3], 30]
- 리스트에서 문자열을 요소로 가짐
  - numbers = [10, 20, "one", 30]
- 중첩 리스트의 내부 리스트의 크기가 일정할 때 이를 2차원 리스트라고 함
  - numbers = [[10, 20], [30, 40], [50, 60]]
  - 외부의 리스트 길이는 3이며, 내부의 리스트는 모두 2개의 요소를 가짐

Nested List

Expression

- numbers = [10, 20, 30]
- letters = [100, "one", numbers]



Nested List

Indexing

✦중첩 리스트에서 인덱스를 이용한 접근 방법

- 인덱스 2개를 사용하여 접근
  - 첫 번째 인덱스는 외부 리스트의 요소를 가리킴
  - 두 번째 인덱스는 내부 리스트의 요소를 가리킴

```
numbers = [10, 20, [1, 2]]
numbers[0] → 10, numbers[1] → 20
numbers[2] → [1, 2, 3]
numbers[2][0] → 1, numbers[2][1] → 2

numbers = [10, 20, "one"]
numbers[2] → "one"
numbers[2][0] → "o", numbers[2][1] → "n"
```

Nested List

Slicing

✦중첩 리스트에서 슬라이싱을 이용한 접근 방법

```
numbers = [10, 20, ["one", "two", "three"]]
numbers[2][1:3] → ["two", "three"]
numbers[1:3][1] → ["one", "two", "three"]
```

인덱스 2(세 번째 요소 리스트)에서 인덱스 1, 2를 가져오기

✦중첩 리스트와 3개의 인덱싱

```
numbers = [10, 20, ["one", "two", "three"]]
numbers[2] → ["one", "two"]
numbers[2][0] → "one", numbers[2][1] → "two"
numbers[2][0][0] → "o", numbers[2][0][1] → "n"
numbers[2][1][1] → "w", numbers[2][1][2] → "o"

numbers = [10, 20, [30, 40, [50, 60]]]
numbers[2][2][2] → 60
```

Practice

Example 8-35

✦5X5 행렬에서 오른쪽 대각선을 기준으로 오른쪽 위의 요소의 합과, 왼쪽 아래 요소의 합 구하기

```
matrix = []; item = 1

for row in range(5):
    rowlist = []
    for col in range(5):
        rowlist.append(item)
        item += 1
    matrix.append(rowlist)

total = 0
for row in range(5):
    for col in range(row, 5):
        total += matrix[row][col]
print("대각선 오른쪽 위 요소의 합 :", total)

total = 0
for row in range(5):
    for col in range(0, row+1):
        total += matrix[row][col]
print("대각선 왼쪽 아래 요소의 합 :", total)
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

```
대각선 오른쪽 위 요소의 합 : 155
대각선 왼쪽 아래 요소의 합 : 235
```

Tuple Type

Overview

✦튜플은 리스트와 같은 형태를 가진 순차 타입의 자료형

- 다양한 자료 형들을 주어진 순서에 따라 저장함
- 저장할 수 있는 모든 종류의 자료를 담을 수 있음

✦리스트에 비해 요소의 내용 변경이 불가능

- 새로운 요소를 추가하거나 삭제와 관련한 메서드는 제공하지 않음
- 정렬도 할 수 없음

✦튜플의 사용 용도

- 리스트가 변경되지 않기를 바라는 경우
- 읽기 전용의 리스트 : 데이터의 보호

✦데이터의 공간이나 크기가 달라지지 않으므로 생성 과정이 간단함

✦리스트에 비해 속도가 빠르며, 프로그램의 성능이 향상됨

Tuple Type

Expression

✦튜플의 생성

- 리스트와 표현 방법이 거의 같으나 '( )' 소괄호로 데이터를 묶어서 표현
- 튜플이름 = (데이터1, 데이터2, 데이터3, ...)
  - 소괄호를 생략하여도 콤마로 데이터를 구분하면 튜플로 처리됨

✦튜플의 초기화

- tuple1 = ()
  - 아무 요소도 존재하지 않은 빈 튜플 생성
- tuple2 = (10,)
  - 요소가 하나인 튜플을 생성
  - 이때 한 개의 요소를 표현할 경우 반드시 뒤에 콤마를 붙임
- tuple3 = (10, 20, 30) or tuple3 = 10, 20, 30
  - 괄호를 생략하더라도 요소 간에 콤마로 구분되면 튜플로 처리

※ 비교

t1 = (1) → 일반 정수
t2 = (1,) → 튜플 생성

Tuple Type

Expression

- `tuple4 = ("korea", "japan", "china")`
  - 문자열을 요소로 하는 튜플을 생성
- `tuple5 = ("korea", 100, 3.14)`
  - 서로 다른 자료형을 요소로 하는 튜플 생성
- `tuple6 = (10, 20, (100, 200), ["korea", "japan"])`
  - 튜플 내에 다른 튜플의 객체나 리스트 등을 요소로 하는 경우

Practice

Example 9-1

```
>>> tuple1 = ()
>>> tuple2 = (10,)
>>> tuple3 = (10)
>>> print(tuple1, tuple2, tuple3)
() (10,) 10
>>> tuple4 = (10, 20, 30)
>>> tuple5 = 10, 20, 30
>>> print(tuple4, tuple5)
(10, 20, 30) (10, 20, 30)
>>> tuple6 = ("korea", 100, 3.14)
>>> tuple7 = (10, 20, ["korea", "japan"])
>>> print(tuple6, tuple7)
('korea', 100, 3.14) (10, 20, ['korea', 'japan'])
```

팩킹(packing)
: 파이썬에서 콤마로 분리된 데이터 값들은 튜플로 인식

Tuple Type

Expression

✦내장 함수 tuple()

- 순차 타입의 자료형을 튜플로 변환하는 내장 함수
- 리스트를 사용하면서 어던 데이터의 변경을 막고자 할 때 사용

```
>>> numbers = tuple()
>>> print(numbers)
()
>>> numbers = [10, 20, 30, 40, 50]
>>> type(numbers)
<class 'list'>
>>> numbers = tuple(numbers)
>>> type(numbers)
<class 'tuple'>
>>> print(numbers)
(10, 20, 30, 40, 50)
>>> numbers = tuple(range(1, 6))
>>> print(numbers)
(1, 2, 3, 4, 5)
```

Tuple Type

Operation

✦튜플의 기본 연산

- + 연산자 : 두 개의 튜플을 합칠 때
- * 연산자 : 하나의 튜플을 반복할 때
- ==, >, >=, <, <= : 두 개의 튜플 요소를 비교
- in 연산자 : 튜플 내에 어떤 요소가 포함되어 있는지 조사

```
>>> t1 = (1, 2, 3)
>>> t2 = (1, 3, 5)
>>> print(t1 + t2)
(1, 2, 3, 1, 3, 5)
>>> print(t1 * 3)
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> t1 > t2
False
>>> 3 in t1
True
```

Tuple Type

Indexing & Slicing

✦인덱싱 (Indexing)

- 튜플도 순차 데이터 타입으로 인덱스를 이용하여 각 요소를 접근함
- 인덱스는 0부터 시작하며, 인덱스 정보를 통해 튜플 요소에 접근
  - 인덱스는 양수 음수 다 가능
- 튜플 요소는 변경을 허용하지 않음

✦슬라이싱 (Slicing)

- 슬라이싱을 통해 튜플에서 일부를 분리하여 더 작은 튜플 생성이 가능

```
>>> numbers = (10, 20, 30, 40, 50)
>>> print(numbers[0], numbers[-1])
10 50
>>> print(numbers[1:4], numbers[1:4:2])
(20, 30, 40) (20, 40)
>>> print(numbers[:2], numbers[3:])
(10, 20) (40, 50)
```

Tuple Type

Nested Tuple

✦튜플 역시 중첩된 구조로 정의할 수 있음

- 튜플의 요소로 문자열, 리스트, 튜플이 가능
- 두 개의 인덱스를 사용하여 접근

```
numbers = (10, "one", (20, 30), [40, 50], ["two", "three"])
numbers[1] → "one",  numbers[2] → (20, 30)
numbers[2:5] → ((20, 30), [40, 50], ['two', 'three'])
numbers[1][1] → "n",
numbers[2][1] → 30
numbers[3][1] → 50
numbers[4][1] → "three"
numbers[4][1][2] → "r"
```

Tuple Type

Delete Item

✦튜플 요소의 삭제

- 튜플의 요소는 수정 뿐만이 아니라 삭제도 불가능
- `del` 키워드를 이용하여 요소를 삭제할 수 없음
- `del` 키워드로 튜플 자체는 삭제 가능

✦튜플의 요소를 삭제하고자 할 경우

- `list()` 함수를 이용하여 리스트로 변환 후 삭제를 수행

✦만약 튜플 내에 있는 리스트의 요소의 경우 수정, 삭제가 가능

✦리스트 내의 튜플의 요소는 수정이 불가능

```
>>> numbers = (10, [20, 30])
>>> numbers[1][0] = 40
>>> print(numbers)
(10, [40, 30])
>>> numbers = [10, (20, 30)]
>>> numbers[1][0] = 40
Traceback (most recent call last):
  File "<pyshell#96>", line 1, in <module>
    numbers[1][0] = 40
TypeError: 'tuple' object does not support item assignment
```

Practice

Example 9-6

```
>>> numbers = (10, 20, 30)
>>> del number[1]  삭제(X)
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    del number[1]
NameError: name 'number' is not defined
>>> numbers = list(numbers)
>>> print(numbers)
[10, 20, 30]
>>> del numbers[1]
>>> numbers = tuple(numbers)
>>> print(numbers)
(10, 30)
>>> del numbers   튜플 자체를 삭제
>>> numbers
Traceback (most recent call last):
  File "<pyshell#90>", line 1, in <module>
    numbers
NameError: name 'numbers' is not defined
```

튜플에 요소를 추가 하거나 수정을 하기 위해서는
다시 리스트로 요소를 생성해야 함

Tuple Type

Delete Item

✦만약 튜플 내에 있는 리스트의 요소의 경우 수정, 삭제가 가능

✦리스트 내의 튜플의 요소는 수정이 불가능

```
>>> numbers = (10, [20, 30])
>>> numbers[1][0] = 40
>>> print(numbers)
(10, [40, 30])
>>> numbers = [10, (20, 30)]
>>> numbers[1][0] = 40
Traceback (most recent call last):
  File "<pyshell#96>", line 1, in <module>
    numbers[1][0] = 40
TypeError: 'tuple' object does not support item assignment
```

Tuple Type

Packing & Unpacking

✦팩킹 (Packing)

- 파이썬에서 콤마(,)로 분리된 데이터 값들을 하나로 묶어 튜플로 인식
- `numbers = 10, 20, 30`
  - 하나의 변수에 여러 개의 정수를 할당
  - 콤마로 분리된 데이터 값들을 하나의 튜플로 인식

✦언팩킹 (Unpacking)

- 하나의 객체로부터 여러 개의 데이터를 가져와 사용
  - 우측의 튜플에서 데이터를 각각 가져와 좌측의 변수들에 할당
- `number1, number2, number3 = 10, 20, 30`
- `letter1, letter2, letter3 = "sea"`
- `number1, number2, number3 = [100, 200, 300]`
- `letter1, letter2 = "python"` → error
- `number1, number2 = [100, 200, 300]` → error

Tuple Type

Packing & Unpacking

✦확장 언팩킹

- 하나의 순차 타입의 객체를 두 가지로 분리
- '*' 를 사용하여 한 개의 값과 나머지 요소들을 리스트로 생성
  - 두 개의 요소에 '*'를 사용할 수 없음

✦사용 예

- number1, *numbers = (10, 20, 30, 40)
  - number1 → 10, numbers → [20, 30, 40]
- *numbers, number1 = (10, 20, 30, 40)
  - numbers → [10, 20, 30], number1 → 40
- letter, *message = "python"
  - letter → 'p', message → ['y', 't', 'h', 'o', 'n']
- *message, letter = "python"
  - message → ['p', 'y', 't', 'h', 'o'], letter → 'n'
- number1, *numbers, *values = (10, 20, 30, 40) → error

Tuple Type

Function & Method

✦튜플 내장 함수

- len(튜플) : 튜플의 길이를 구함
- max(튜플) : 튜플의 요소 중 가장 큰 값을 구함
- min(튜플) : 튜플의 요소 중 가장 작은 값을 구함

✦튜플 메서드

- 튜플.count(찾을 요소)
  - 리스트 내의 요소들 중 특정 데이터가 몇 개나 있는지 확인
- 튜플.index(찾을 요소, 찾을 위치)
  - 인수로 지정한 값이 튜플에서 처음 나타나는 인덱스를 반환

```
>>> score = (95, 88, 92, 100, 70)
>>> print(len(score), max(score), min(score))
5 100 70
>>> numbers = (10, 20, 30, 10, 40, 10, 50)
>>> numbers.count(10)
3
>>> numbers.index(10)
0
```

Dictionary Type

Overview

✦사전 타입은 비 순서타입의 자료형

- 데이터의 저장 순서가 없음
- 인덱싱과 슬라이싱에 의한 접근이 불가능

✦키에 의한 매핑방식으로 접근

- 사전의 요소는 키와 값의 쌍으로 구성되어 있는 하나의 집합적인 구조
- 키를 이용하여 데이터를 검색

✦해싱(Hashing)방식으로 데이터에 접근하므로 탐색 속도가 빠름

- 해싱 : 키를 이용하여 데이터가 저장된 위치의 주소를 계산하는 방법

✦키는 숫자나 문자열을 비롯하여 어떤 자료형이라도 사용 가능하므로 데이터를 하나로 묶는데 매우 효과적

- 리스트와 사전은 제외

Dictionary Type

Expression

✦사전의 형태

- 전체를 대괄호 '{ }' (중괄호)로 묶어 시작과 끝을 표시함
- 하나의 요소는 키와 값의 쌍으로 표현, 각 요소는 콤마로 구분
- 사전의 키들은 유일한 값이어야 하며, 순서는 정해져 있지 않음
- 사전이름 = {키1:값1, 키2:값2, 키3:값3, ... }

키와 값의 한 쌍

데이터의 순서는 관계 없음
데이터의 표시가 되는 이름을 붙이고 저장
Value Key

key	value
name	kim
phone	101234
birth	1224

값에는 리스트도 포함될 수 있음

- person = {"name":"kim", "phone":101234, "birth":1224}
- print(person)
  - 출력 순서는 정해진 것이 아니므로 다를 수 있음

Dictionary Type

Expression

✦사전의 초기화

- `numbers = {}`
  - 빈 사전을 생성
- `numbers = {1:"one", 2:"two", 3:"three", 4:"four"}`
  - 정수로 구성된 키와 각 키에 해당하는 값들을 모두 문자열로 구성
- `numbers = {"one":1, "two":2, "three":3, "four":4}`
  - 문자열로 구성된 키와 각 키에 해당하는 값들을 모두 정수로 구성
- `numbers = {1:"one", 2:"two", "three":3, "four":4}`
  - 키들은 정수와 문자열이 같이 사용되어 있고, 각 키에 해당하는 값들도 정수와 문자열이 같이 구성
- `numbers = {"name":["홍길동", "박지성"], "score":(100, 95)}`
  - 각 키에 해당하는 값들을 리스트와 튜플로 구성

Dictionary Type

Expression

✦dict() 함수의 사용

- 사전 타입의 자료형 객체를 반환하는 함수
- dict() : 함수에 인수를 지정하지 않을 경우 빈 사전이 생성
- dict(리스트형의 사전) 리스트형의 사전을 사전 타입으로 변환
  - 리스트형의 사전 : 튜플을 요소로 하는 리스트로 각 튜플은 키와 값으로 구성됨
- dict([("one", 1), ("two", 2), ("three", 3)])
  - {'one': 1, 'two': 2, 'three': 3} 사전이 생성
- numbers = dict(one=1, two=2) → {'one': 1, 'two': 2}
  - 함수의 인수에 key=item 쌍을 나열하여 사전을 정의
- numbers = dict(1="one", 2="two") → error
  - Key는 숫자가 올 수 없으며, 문자만 가능하다.

Dictionary Type

Data Access

✦사전 요소의 데이터 접근

- 사전에서는 인덱스를 사용하여 데이터에 접근할 수 없음
- 사전의 각 요소에 해당하는 값은 키를 통해 접근함
  - 값에 해당하는 키를 인덱스처럼 사용

```
numbers = {1:"one", 2:"two", 3:"three", 4:"four"}
numbers[1] → "one"  키에 해당하는 값을 찾음
numbers[2] → "two"
```

```
numbers = {"one":1, "two":2, "three":3, "four":4}
numbers["one"] → 1
numbers["two"] → 2
```

✦사전 요소의 변경

- 사전의 각 요소에 해당하는 값은 키를 통해 값을 변경할 수 있음

Dictionary Type

Data Access

✦사전 요소 변경의 예

- `numbers = {1:"one", 2:"two", 3:"three"}`
- `numbers[1] = "hana"` → 값 수정
  - 키가 사전에 이미 존재할 경우 값을 대입하는 형태로 값을 변경할 수 있음
- `numbers[4] = "four"` → 값 추가
  - 키가 사전에 등록되어 있지 않을 경우 새로운 요소로 사전에 추가
  - 키가 다르더라도 동일한 값은 여러 개 존재할 수 있음
- `numbers[5] = [10, 20, 30]`
  - 값은 리스트, 튜플과 같은 순차 타입의 객체도 가능
- `numbers["six"] = "여섯"`
  - 새로운 키는 변경이 불가능한 자료형으로 정수, 문자열 등을 사용할 수 있음
- `numbers[(1,2,3)] = "triple"` → 추가
  - 튜플도 키가 될 수 있지만 리스트와 사전은 불가능
- `del numbers[4]` → 키에 해당하는 사전의 요소 삭제

Dictionary Method

Type Casting

✦사전 요소의 리스트 반환

- 사전의 키나 값들을 순서에 의해 처리하기 위해서 리스트나 튜플로 변환
- 주로 for문을 사용하여 사전의 키나 값들을 차례로 가져옴

✦주요 메서드

- 사전.keys()
  - 사전의 요소에서 키 만을 따로 모아서 뷰 객체로 생성
    - 뷰(View) 객체는 데이터들의 목록을 의미하는 객체
    - 생성된 뷰 객체는 리스트나 튜플로 변환하여 사용함
- 사전.values()
  - 사전의 요소에서 키에 대응하는 값 만을 따로 모아서 뷰 객체로 생성
- 사전.items()
  - 사전의 요소에서 키와 값을 튜플로 구성한 항을 모아서 뷰 객체로 생성
  - 프로그램에서 두 개의 쌍이 모두 필요할 경우 이 메서드를 사용