CHAPTER 7

객체복사, 순회 연산자와 함수

Copying Objects and Iterating Operations

박 진 수 교수
서울대학교•경영대학
jinsoo@snu.ac.kr

학습 목차

- ☑ 객체 복사
- ☑ 순회형 연산자와 함수

객체 복사

Copying Objects

객체 참조

객체 참조(object reference)란?

객체를 쉽게 불러오기 위해 컴퓨터 메모리 어딘가에 있는 개별 객체의 위치를 참조하는 것

주로 **변수**를 통해 객체 참조를 한다

객체 참조의 값이 다르다는 것은 어떤 의미인가?

객체 비교

두 객체의 객체 참조 값이 다르면 객체가 서로 다른 메모리 공간에 있기 때문에 비록 같은 값이어도 다른 객체

이는 마치 사람의 이름이 같아도 서로 다른 주소에 산다면 다른 사람인 것에 비유할 수 있다

```
x = 'stay hungry, stay foolish'
y = 'stay hungry, stay foolish'

print(x)
print(y)
```

```
stay hungry, stay foolish
stay hungry, stay foolish
```

두 문자열의 값이 같은지 확인해보자

객체 비교

둘의 값은 같은가?

```
x == y
```

이처럼 두 객체의 값이 같은지 확인하는 연산자로는 ==가 있다

그렇다면 이 두 문자열이
서로 같은 객체를 참조하는지 확인하는 방법은 없을까?

항등 연산자

항등 연산자(identity operator)란?

좌변의 변수(객체 참조)가 우변의 변수(객체 참조)와 같은/다른 객체를 참조하면 `True`/`False`를 반환

`is` / `is not`

```
# 둘은 같은 객체를 참조하고 있는가?
x is y
```

x 와 ***y*** 가 같은 객체를 참조하면(즉, 같은 메모리 공간에 있으면) '참(`True`)' 을 반환하고, 아니면 '거짓(`False`)'을 반환

예시 : 항등 연산자

이번에는 새로운 변수 ***z***에 ***x***를 할당해보자

```
# z에 x를 할당한다.
z = x
print(z)
```

```
# z와 y의 값은 같은가?
z == y
```

```
# z와 y는 같은 객체를 참조하고 있는가?
z is y
```

```
# z와 x는 같은 객체를 참조하고 있는가?
z is x
```

항등 연산자

- 언제 사용?
  - 주로 객체가 비어있는지(None), 즉 변수에 객체가 할당되어 있는지 여부를 확인할 때 사용
  - 모든 파이썬 변수는 사실상 객체 참조이기 때문에 어떤 경우에는 각각의 변수가 동일한 객체를 가리키고 있는지 확인해야 할 필요가 있는데 이럴 때도 사용
  - 대부분의 경우 사용자는 데이터 값을 비교하고 싶어하기 때문에 항등 연산자 `is`를 사용하는 것이 필요 없지만...
  - 항등 연산자의 **장점**은 객체의 메모리 주소 값만을 비교하기 때문에 처리 속도가 더 빠르다는 장점이 있다
    - 메모리 주소가 같다는 것은 같은 객체라는 뜻

복사하는 방법

1	할당(객체 참조)
2	얕은 복사(shallow copy)
3	깊은 복사(deep copy)

첫 번째 방법 : 할당(객체 참조)

할당 연산자(=)

할당 연산자는 실제로 해당 객체 자체를 '복사'하는 것이 아니라 해당 객체에 대한 객체 참조를 복사

문제는 **가변자료형**이다

가변자료형은 값을 변경할 수 있기 때문에, 한쪽의 값을 변경하면 다른 쪽도 변경한 값을 참조하게 된다

```
x = [1, 2, 3]
y = [1, 2, 3]
print(x)
print(y)
x is y
```

```
z = x
print(z)
```

```
z.append('a')
print(z)
print(x)
print(y)
```

```
z is x
z is y
```

두 번째 방법 : 얕은 복사

- 얕은 복사(shallow copy)란?
  - 원본 객체가 가지고 있는 모든 객체의 객체 참조를 복사해서 원본 객체와는 별도로 새로운 복사본(객체)를 만들어 반환
    - 원본 객체와 별도로 개별 복사본을 생성하지만, 원본과 복사본 **둘 다 같은 객체를 참조**하고 있다
    - 얕은 복사를 하면 참조하는 객체를 서로 공유하기 때문에 별도로 객체 자체를 따로 복사하는 것 보다 속도도 빠르고 메모리 공간도 적게 사용하기 때문에 더 효율적임
  - 불변자료형(immutable data types)의 경우(예 : 문자열, 숫자 등)
    - 값을 바꿀 수 없기 때문에 일반적인 복사와 같은 기능을 수행(다만 얕은 복사가 더 효율적임)
  - 가변자료형(mutable data types)의 경우(예 : 리스트, 딕셔너리 등)
    - 가변자료형이 불변자료형만 객체로 가지고 있으면,
      - 값을 바꿀 수 없기 때문에 일반적인 복사와 같은 기능을 수행(다만 얕은 복사가 더 효율적임)
      - 예) 리스트가 또 다른 리스트나 딕셔너리를 객체로 가지고 있지 않은 경우
    - 가변자료형이 가변자료형을 객체로 가지고 있으면,
      - 원본과 복사본 둘 다 같은 객체 참조를 하고 있기 때문에 둘 중 하나의 값이 바뀌면 나머지도 바뀐 내용을 보게된다
      - 예) 리스트가 또 다른 리스트나 딕셔너리를 객체로 가지고 있는 경우

얕은 복사 방법

- 분할([:])
  - 분할은 복사하려는 객체와 별도로 개별 복사본을 생성
- 복합자료형 copy() 메소드
  - *list*.copy()
  - *set*.copy()
  - *dict*.copy()
- 복합자료형 생성자(constructors)
  - list(순회형)
  - set(순회형)
  - dict(순회형)
- copy() 함수
  - copy 모듈에 있는 copy() 함수 사용
    - copy.copy()

얕은 복사 : 분할([:])

```
x = [1, 2, 3]
y = x          # 할당 복사(객체 참조)
y is x
```

```
x[-1] = 'a'
print(x)
print(y)
```

```
z = x[:]       # 분할 연산자[:]로 얕은 복사
z is x
print(z)
print(x)
```

```
z[0] = 'b'
print(z)
print(x)
print(y)
```

얕은 복사 : 복합자료형 copy() 메소드

```
x = [1, 2, 3]
y = x          # 할당 복사(객체 참조)
y is x
```

```
x[-1] = 'a'
print(x)
print(y)
```

```
z = x.copy() # copy() 메소드로 얕은 복사
z is x
print(z)
print(x)
```

```
z[0] = 'b'
print(z)
print(x)
print(y)
```

얕은 복사 : 복합자료형 생성자

```
x = [1, 2, 3]
y = x           # 할당 복사(객체 참조)
y is x
```

```
x[-1] = 'a'
print(x)
print(y)
```

```
z = list(x)    # 리스트 생성자로 얕은 복사
z is x
print(z)
print(x)
```

```
z[0] = 'b'
print(z)
print(x)
print(y)
```

얕은 복사 : copy.copy() 함수

```
x = [1, 2, 3]
y = x                  # 할당 복사(객체 참조)
y is x
```

```
x[-1] = 'a'
print(x)
print(y)
```

```
import copy          # copy 모듈을 불러온다
z = copy.copy(x)     # copy() 함수로 얕은 복사
z is x
print(z)
print(x)
```

```
z[0] = 'b'
print(z)
print(x)
print(y)
```

중첩 가변자료형의 얕은 복사

- 문제점
  - 얕은 복사의 경우 원본과 복사본 둘 다 같은 객체를 참조를 하고 있기 때문에 가변자료형이 또 다른 가변자료형을 가지고 있는 중첩 복합자료형의 경우 원본과 복사본 둘 중 하나의 값이 바뀌면 나머지도 영향을 받는다
    - 중첩(nested) 복합자료형 : 복합자료형이 또 다른 복합자료형을 객체로 가지고 있는 자료 구조

```
x = [1, 2, ['x', 'y', 'z']]
y = x[:]      # 얕은 복사
x == y
x is y
x[0] is y[0]
```

```
x[0] = 'a'
print(x)
print(y)
```

```
x[0] is y[0]
```

```
x[-1] is y[-1]
```

```
y[-1][-1] = 'b'
print(y)
print(x)
```

```
x[-1][-1] is y[-1][-1]
```

세 번째 방법 : 깊은 복사

깊은 복사(deep copy)란?

원본 객체에 있는 모든 가변자료형 **객체 자체를 별도로 복사**해서 별도의 복사본을 생성 —> 따라서 복사본은 새로 복사한 독립된 객체를 참조

copy 모듈에 있는 deepcopy() 함수 사용

```
x = [1, 2, ['x', 'y', 'z']]

import copy
y = copy.deepcopy(x)  # 깊은 복사
x == y
x is y
x[0] is y[0]
```

```
x[0] = 'a'
print(x)
print(y)
```

```
x[0] is y[0]
```

```
x[-1] is y[-1]
```

```
y[-1][-1] = 'b'
print(y)
print(x)
x[-1][-1] is y[-1][-1]
```

순회형 연산자와 함수

Iterating Operators and Functions

순회형 연산자와 함수

문법	설명
`s + t`	▸ 시퀀스형 `s`와 `t`를 합친 시퀀스형 자료를 반환
`s * n`	▸ 시퀀스형 `s`를 `n` 횟수만큼 반복한 시퀀스형 자료를 반환
`x in i`	▸ 순회형 `i`에 객체 `x`가 있으면 참(`True`)을 반환 ▸ `not in`은 순회형 `i`에 객체 `x`가 없으면 참(`True`)을 반환
`len(x)`	▸ `x`의 길이를 반환 – `x`가 복합자료형이면 객체의 총 개수를 반환 – `x`가 문자열이면 문자의 개수를 반환
`all(i)`	▸ 순회형 `i`의 모든 객체가 '참'일 때만 참(`True`)을 반환
`any(i)`	▸ 순회형 `i`의 한 객체라도 '참'이면 참(`True`)을 반환
`max(i, key)`	▸ 순회형 `i`의 객체 중 가장 큰 값을 가진 객체를 반환 ▸ `key` 전달인자가 주어지면 전달인자 함수로 처리한 결괏값 중 가장 큰 값을 가진 객체를 반환
`min(i, key)`	▸ 순회형 `i`의 객체 중 가장 작은 값을 가진 객체를 반환 ▸ `key` 전달인자가 주어지면 전달인자 함수로 처리한 결괏값 중 가장 작은 값을 가진 객체를 반환

순회형 연산자와 함수

문법	설명
`sum(i[, start])`	▸ 순회형 *i*의 모든 객체를 더한 값을 반환 ▸ *start* 전달인자가 주어지지 않으면 기본값은 0 ▸ *start* 전달인자가 주어지면 *start* 전달인자와 순회형 *i*의 객체들의 전체 합을 반환 ▸ *i*에 문자열이 포함되어 있으면 TypeError 가 발생
`reversed(seq)`	▸ 시퀀스형 *seq*의 주어진 객체 순서를 역순으로 순회하는 순회자(iterator)를 반환
`sorted(i, key=None, reverse=False)`	▸ 순회형 *i*의 객체를 정렬하는 방식의 순서로 바꾼 후 리스트로 반환 ▸ *key* 전달인자를 사용하면 DSU(Decorate, Sort, Undecorated) 정렬이 가능 ▸ *reverse* 전달인자가 참(True)이면 정렬이 역순으로 이루어진다
`range(start, stop, step)`	▸ 정수 순회자를 반환 – 한 개의 전달인자(*stop*)가 주어지면 순회자는 0부터 *stop* - 1까지의 정수를 반환 – 두 개의 전달인자(*start*, *stop*)가 주어지면 순회자는 *start*부터 *stop* - 1까지의 정수를 반환 – 세 개의 전달인자(*start*, *stop*, *step*)가 주어지면 순회자는 *start*부터 *step*만큼의 간격을 두고 *stop* - 1까지의 정수를 반환
`enumerate(i, start=0)`	▸ 보통 for ... in 순환문과 함께 사용하며 (인덱스, 객체)의 튜플 쌍으로 순회할 수 있는 열거형(enumerate) 객체를 반환 ▸ *start* 전달인자가 주어지지 않으면 기본 값은 0 ▸ *start* 전달인자가 주어지면 *start* 전달인자부터 인덱스가 시작
`zip(i1,...,iN)`	▸ 순회자 *i1, ... ,iN* 을 사용하여 튜플 순회자를 반환

순회형 연산자

s + *t*

시퀀스형 *s*와 *t*를 합친 시퀀스형 자료를 반환

s * *n*

시퀀스형 *s*를 *n* 횟수만큼 반복한 시퀀스형 자료를 반환

x in *i*

순회형 *i* 에 객체 *x* 가 있으면 참(True)을 반환

x not in *i*

순회형 *i* 에 객체 *x* 가 없으면 참(True)을 반환

```
[1, 2, 3] + ['a', 'b', 'c']
```

```
(1, 2, 3) * 5
```

```
'a' in {'a', 'b', 'c'}
```

```
'x' not in {'a', 'b', 'c'}
```

순회형 함수 : 질의 함수

`len(x)`

*x*의 길이를 반환

- *x*가 복합자료형이면 객체의 총 개수를 반환
- *x*가 문자열이면 문자의 개수를 반환

`all(i)`

순회형 `i`의 모든 객체가 '참'일 때만 참(`True`)을 반환

`any(i)`

순회형 `i`의 한 객체라도 '참'이면 참(`True`)을 반환

```
len([1, 2, 3])
```

```
len('Python')
```

```
x = [7, -5, 8, 3, 9, 0]
all(x)
any(x)
```

```
any([False, 0, 0.0, None, '', [], (), set(), {}])
```

```
all([-15, 1.34, [1], (1,), set('a'), {None: 'NaN'}, ''])
```

순회형 함수 : 연산 함수

max(*i*, *key*)

순회형 *i*의 객체 중 가장 큰 값을 가진 객체를 반환

- *key* 전달인자가 주어지면 전달인자 함수로 처리한 결괏값 중 가장 큰 값을 가진 객체를 반환

min(*i*, *key*)

순회형 *i*의 객체 중 가장 작은 값을 가진 객체를 반환

- *key* 전달인자가 주어지면 전달인자 함수로 처리한 결괏값 중 가장 작은 값을 가진 객체를 반환

sum(*i*[, *start*])

순회형 *i*의 모든 객체를 더한 값을 반환

- *start* 전달인자가 주어지지 않으면 기본값은 0
- *start* 전달인자가 주어지면 *start* 전달인자와 순회형 *i*의 객체들의 전체 합을 반환
- *i*에 문자열이 포함되어 있으면 TypeError 가 발생

```
x = [7, -5, 8, 3, 9]
min(x)
max(x)
min(x, key=abs)
```

```
s = ['a', 'B', 'c', 'd', 'E']
max(s)
min(s)
max(s, key=str.lower)
```

```
x = [7, -5, 8, 3, 9]
sum(x)
sum(x, 9)
```

순회형 함수 : 정렬 함수

reversed(*seq*)

시퀀스형 *seq*의 주어진 객체 순서를 역순으로 순회하는 순회자(iterator)를 반환

```
L = [7, -5, 8, 3, 9]
reversed(L)
type(reversed(L))
list(reversed(L))
```

sorted(*i*, *key=None*, *reverse=False*)

순회형 *i*의 객체를 정렬하는 방식의 순서로 바꾼 후 리스트로 반환

- *key* 전달인자를 사용하면 DSU(Decorate, Sort, Undecorated) 정렬이 가능
- *reverse* 전달인자가 참(True)이면 정렬이 역순으로 이루어진다

```
L = [7, -5, 8, 3, 9]
sorted(L)
sorted(L, reverse=True)
sorted(L, key=abs)
sorted(L, key=abs, reverse=True)
```

순환문과 함께 사용하는 유용한 함수

다음 세 함수는 8장의 순환문에서 상세히 다룬다

```
range(start, stop, step)
```

```
enumerate(i, start=0)
```

```
zip(i1,...,iN)
```