파이썬 프로그래밍

# 클래스와 연산자 중복 정의


한국기술교육대학교
온라인평생교육원

## ▣ 시퀀스/매핑 자료형의 연산자 중복

- 클래스를 개발할 때 다음 메소드들을 적절하게 구현하면 자신만의 시퀀스 자료형을 만들 수 있음
- 변경불가능한 (Immutable) 시퀀스 자료형 및 매핑 자료형을 위해 구현이 필요한 메소드

| 메소드 | 연산자 |
|---|---|
| __len__(self) | len() |
| __contains__(self, item) | item in self |
| __getItem__(self, key) | self[key] |
| __setItem__(self, key, value) | self[key] = value |
| __delItem__(self, key) | del self(key) |

▪item 인자가 contains의 두 번째 item 인자로 들어옴
▪itme이 self 안에 존재하는지 알아보는 멤버쉽 테스트 연산자와 매핑
▪[key] → 인덱스 연산
▪매핑 자료형이면 검색 연산이 됨
▪튜플, 문자열 같은 변경 불가능한 것 → setitem 구현 X

## ▣ 시퀀스/매핑 자료형의 연산자 중복

### 1. 인덱싱

- len(s1) --> s1.__len__() 메소드 호출
- sl[4] --> s1.__getitem__(4) 호출
- IndexError
  - 시퀀스 자료형이 범위를 벗어난 인덱스 참조 요구시에 발생됨
  - 리스트, 튜플, 문자열등에서도 동일한 조건에서 발생됨

▪sequare → 어떤 숫자의 제곱에 해당하는 것을 반환
▪s1 = Square(10) → 10은 end에, s1은 10(end값)이 할당되어 있음
▪len() → __len__()
▪s1[1] → __getitem__(self, k)
▪len() → __len__()
▪s1[1] → __getitem__(self, k)

## ▣ 시퀀스/매핑 자료형의 연산자 중복

### 1. 인덱싱

```
class Square:
    def __init__(self, end):
        self.end = end
    def __len__(self):
        return self.end
    def __getitem__(self, k):
        if k < 0 or self.end <= k:
            raise IndexError, k
        return k * k

s1 = Square(10)
print len(s1) # s1.__len__()
print s1[1] #s1.__getitem__(1)
print s1[4]
print s1[20]
```

```
10
1
16
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-3-78c6c0117c4f> in <module>()
     13 print s1[1] #s1.__getitem__(1)
     14 print s1[4]
---> 15 print s1[20]

<ipython-input-3-78c6c0117c4f> in __getitem__(self, k)
      6     def __getitem__(self, k):
      7         if k < 0 or self.end <= k:
----> 8             raise IndexError, k
      9         return k * k
     10

IndexError: 20
```

## ▣ 시퀀스/매핑 자료형의 연산자 중복

### 1. 인덱싱

- 다음 for 문은 s1에 대해 __getitem()__ 메소드를 0부터 호출하여 IndexError가 발생하면 루프를 중단한다.

```
for x in s1:
    print x,
```

```
0 1 4 9 16 25 36 49 64 81
```

▪s1 안에서 객체 x를 꺼내는 것
▪for x in s1: → __getitem__ 호출하여 나온 결과값을 하나씩 x에 삽입
▪인덱스 error가 발생하면 for ~ in 구문 멈춤
▪if 구문을 만족시키지 않으면 계속 진행
▪k가 10이면 if 절을 만족하여 IndexError
▪__getitem__ 메소드가 호출
▪getitem에 반드시 error를 발생시켜 for~in 구문 마침

## ▣ 시퀀스/매핑 자료형의 연산자 중복

### 1. 인덱싱

- __getitem__() 메소드가 정의되어 있다면 다른 시퀀스 자료형으로 변환이 가능

```
print list(s1)
print tuple(s1)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
(0, 1, 4, 9, 16, 25, 36, 49, 64, 81)
```

▪list, tuple → __getitem__을 호출

- 위에서 알 수 있듯이 파이썬은 내장 자료형과 개발자가 정의한 자료형에 대해 일관된 연산 적용이 가능
  - 파이썬 언어의 장점: 일관된 코딩 스타일 유지

▪s1 객체는 Square(10) 객체이지만 여러 연산자, 메소드 사용 가능

## ▣ 시퀀스/매핑 자료형의 연산자 중복

### 2. 매핑 자료형의 연산자 중복

```
class MyDict:
    def __init__(self, d = None):
        if d == None: d = {}
        self.d = d
    def __getitem__(self, k): #key
        return self.d[k]
    def __setitem__(self, k, v):
        self.d[k] = v
    def __len__(self):
        return len(self.d)

m = MyDict()            #__init__호출
m['day'] = 'light'      #__setitem__호출
m['night'] = 'darkness' #__setitem__호출
print m
print m['day']          #__getitem__호출
print m['night']        #__getitem__호출
print len(m)            #__len__호출
```

```
<__main__.MyDict instance at 0x10bb37638>
light
darkness
2
```

## ▣ 시퀀스/매핑 자료형의 연산자 중복

### 2. 매핑 자료형의 연산자 중복

▪MyDic : 스스로 dictionary를 정의
▪none 이 디폴트 인수 → 기본인수가 none
▪어떤 값이 존재하면 self.d에 들어감 → 만들려는 인스턴스에 식별자 생성
▪setitem의 두 번째 인자 k : key 값, 세 번째 인자 v : value
▪len(self.d) → self가 들고 있는 d의 길이
▪m이 현재 가지고 있는 d에 setitem이 불려짐
▪MyDict() 인자가 없으므로 빈 문자열이 들어감
▪k → 'day', v → 'light'
▪사전 안에는 아이템이 2개 존재
▪len(m) → m이 가지고 있는 __len__ 호출 → 가지고 있는 사전의 길이

## ▣ 시퀀스/매핑 자료형의 연산자 중복

### 2. 매핑 자료형의 연산자 중복

```
class MyDict:
    def __init__(self, d=None):
        if d == None: d = {}
        self.d = d
    def __getitem__(self, k):
        return self.d[k]
    def __setitem__(self, k, v):
        self.d[k] = v
    def __len__(self):
        return len(self.d)
    def keys(self):
        return self.d.keys()
    def values(self):
        return self.d.values()
    def items(self):
        return self.d.items()

m = MyDict({'one':1, 'two':2, 'three':3})
print m.keys()
print m.values()
print m.items()
```

```
['three', 'two', 'one']
[3, 2, 1]
[('three', 3), ('two', 2), ('one', 1)]
```

▪d에는 사전 위치가 할당됨