파이썬 프로그래밍

상속과 다형성

한국기술교육대학교
온라인평생교육원

▣ 내장 자료형과 클래스의 통일

1. 리스트 서브 클래스 만들기

- 내장 자료형(list, dict, tuple, string)을 상속하여 사용자 클래스를 정의하는 것
  - 내장 자료형과 사용자 자료형의 차이를 없에고 통일된 관점으로 모든 객체를 다룰 수 있는 방안
- 클래스 정의는 새로운 자료형의 정의임

```
a = list()
print a
print dir(a)
```

```
<Person gslee 5284>
<Person kslee 5224>
```

- 상속을 받을 때 list를 상속 받음
- a = [] == a = list()
- list를 클래스 명으로 하여 ()를 붙여 인스턴스를 만듦
- 기본적으로 a는 일반적인 list
- 소문자 list가 클래스 명

▣ 내장 자료형과 클래스의 통일

1. 리스트 서브 클래스 만들기

- 아래 예제는 내장 자료형인 list를 상속하여 뺄셈 연산(-)을 추가함

```
class MyList(list):
    def __sub__(self, other):   # '-' 연산자 중복 함수 정의
        for x in other:
            if x in self:
                self.remove(x)     # 각 항목을 하나씩 삭제한다.
        return self

L = MyList([1, 2, 3, 'spam', 4, 5])
print L
print

L = L - ['spam', 4]
print L
```

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

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

- - 가 __sub__에 대응됨
- other → ['spam', 4]
- 상속받는 클래스의 객체는 부모 클래스의 타입과 동일
- L은 리스트이면서 1,2,3,'spam',4,5 원소를 가짐
- print L에 대응되는 __str__이 없음
- 하지만 클래스 list가 가지고 있는 _str__ 활용
- 내장자료형인 list에는 __sub__은 정의가 되어 있지 않음
- 따라서 MyList에서는 생성자를 더 해 더 풍부하게 만듦
- self → list
- L에는 append가 없지만 기존 list는 가지고 있으므로 수행됨

▣ 내장 자료형과 클래스의 통일

1. 리스트 서브 클래스 만들기

1) Stack 클래스 정의 예
- 슈퍼 클래스로 list 클래스를 지닌다.
- 즉, list 클래스를 확장하여 Stack 클래스를 정의함

```
class Stack(list):  # 클래스 정의
    push = list.append

s = Stack()         # 인스턴스 생성

s.push(4)
s.push(5)
print s
print

s = Stack([1,2,3])
s.push(4)
s.push(5)
print s
print

print s.pop()       # 슈퍼 클래스인 리스트 클래스의 pop() 메소드 호출
print s.pop()
print s
```

```
[4, 5]

[1, 2, 3, 4, 5]

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

▣ 내장 자료형과 클래스의 통일

1. 리스트 서브 클래스 만들기

- Stack 에는 push와 pop 연산자 존재
- push를 따로 정의 X → list가 가지고 있는 append를 공유
- print s → Stack 안 str 없으므로 list의 str 활용
- pop은 Stack에 없으나 list에 존재하여 활용 가능

▣ 내장 자료형과 클래스의 통일

1. 리스트 서브 클래스 만들기

2) Queue 클래스 정의 예
- 슈퍼 클래스로 역시 list를 지닌다.
- 즉, list 클래스를 확장하여 Queue 클래스를 정의함

```
class Queue(list):
    enqueue = list.append
    def dequeue(self):
        return self.pop(0)

q = Queue()
q.enqueue(1)      # 데이터 추가
q.enqueue(2)
print q

print q.dequeue() # 데이터 꺼내기
print q.dequeue()
```

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

- enqueue() : 리스트 맨 뒤에 원소 추가 = append
- dequeue() : 리스트 맨 앞에서 원소 꺼냄

▣ 내장 자료형과 클래스의 통일

2. 사전 서브 클래스 만들기

```
a = dict()
print a
print dir(a)
```

```
{}
['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear',
'copy', 'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'iterkeys', 'itervalues', 'keys',
'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']
```

- dict 는 클래스로, 생성자를 호출하여 a에 넣어 사전 만듦
- a에는 사전에서 많이 호출할 수 있는 메소드들 포함

▣ 내장 자료형과 클래스의 통일

2. 사전 서브 클래스 만들기

- 아래 예제는 keys() 메소드를 정렬된 키값 리스트를 반환하도록 재정의한다.

```
class MyDict(dict):
    def keys(self):
        K = dict.keys(self) # 언바운드 메소드 호출 --> K = self.keys() 라고 호출하면
                                무한 재귀 호출
        K.sort()
        return K

d = MyDict({'one':1, 'two':2, 'three':3})
print d.keys()
print

d2 = {'one':1, 'two':2, 'three':3}
print d2.keys()
```

```
['one', 'three', 'two']

['three', 'two', 'one']
```

- keys는 기존 dict에 존재하지만 재정의하여 사용
- dic.keys() → 클래스의 keys를 부르므로 언바운드 매소드
- self.keys를 부르면 dict의 keys가 아닌 구현하고 있는 keys를 매핑
- 무한루프로 무한 재귀 호출 → error 발생