파이썬 프로그래밍

클래스와 연산자 중복 정의

한국기술교육대학교
온라인평생교육원

▣ 연산자 중복

1. 수치 연산자 중복

- 직접 정의하는 클래스 인스턴스에 연산자를 적용하기 위하여 미리 약속되어 있는 메소드들을 정의

메소드(Method)	연산자(Operator)	인스턴스 o에 대한 사용 예
__add__(self, B)	+ (이항)	o + B, o += B
__sub__(self, B)	- (이항)	o - B, o -= B
__mul__(self, B)	*	o * B, o *= B
__div__(self, B)	/	o / B, o /= B
__floordiv__(self, B)	//	o // B, o //= B
__mod__(self, B)	%	o % B, o %= B
__divmod__(self, B)	divmod()	divmod(o, B)
__pow__(self, B)	pow(), **	pow(o, B), o ** B
__lshift__(self, B)	<<	o << B, o <<= B
__rshift__(self, B)	>>	o >> B, o >>= B
__and__(self, B)	&	o & B, o &= B
__xor__(self, B)	^	o ^ B, o ^= B
__or__(self, B)	|	o | B, o |= B
__neg__(self)	- (단항)	-A
__abs__(self)	abs()	abs(o)
__pos__(self)	+ (단항)	+o
__invert__(self)	~	~o

▣ 연산자 중복

1. 수치 연산자 중복

- ▪클래스를 직접 정의
- ▪클래스에서 객체를 인스턴스화하여 활용
- ▪이러한 객체에 연산을 적용 → 어떻게 해야 할까?
- ▪인스턴스 메소드 → 첫 번째 인자에는 self가 옴
- ▪+ 연산자 사용하면 자동으로 add 메소드 불러짐
- ▪+=, -= 처럼 확장연산자 사용 가능

▣ 연산자 중복

1. 수치 연산자 중복

```
class Integer:
    def __init__(self, i):
        self.i = i
    def __str__(self):
        return str(self.i)
    def __add__(self, other):
        return self.i + other

i = Integer(10)
print i
print str(i)

print
i = i + 10
print i

print
i += 10
print i
```

```
10
10

20

30
```

▣ 연산자 중복

1. 수치 연산자 중복

▪__str__은 객체를 프린트할 때 호출됨
▪i = i + 10을 했을 때 add 메소드 호출됨
▪10 → 두 번째 인자 other에 들어감
▪str(i)의 i는 객체, self.i의 i는 i 식별자 값
▪self.i + other → self.i 는 정수, other도 정수 → 정수가 반환됨
▪print i는 객체가 아닌 정수를 출력
▪print i += 10에서 +=는 add를 호출하지 않음
▪print self만 하게 되면 정수 i 가 아닌 객체 i
▪객체를 프린트하니까 str이 호출되어 문자화가 된 self.i 값 출력

▣ 연산자 중복

1. 수치 연산자 중복

```
class MyString:
    def __init__(self, str):
        self.str = str
    def __div__(self, sep): # 나누기 연산자 /가 사용되었을 때 호출되는 함수
        return self.str.split(sep) # 문자열 self.str을 sep를 기준으로 분리

m = MyString("abcd_abcd_abcd")
print m / "_"
print m / "_a"

print
print m.__div__("_")
```

```
['abcd', 'abcd', 'abcd']
['abcd', 'bcd', 'bcd']

['abcd', 'abcd', 'abcd']
```

▪self.str.split(sep) → self 객체가 가지고 있는 str을 sep으로 분리

▪'_'를 가지고 있는 문자열을 찾아 sep에 집어 넣음

▪특수 메소드는 / 기호 또는 직접 이름을 넣어 호출 가능

▣ 연산자 중복

1. 수치 연산자 중복

- 연산자 왼쪽에 피연산자, 연산자 오른쪽에 객체가 오는 경우
  - 메소드 이름 앞에 r이 추가된 메소드 정의

```
class MyString:
    def __init__(self, str):
        self.str = str
    def __div__(self, sep):
        return str.split(self.str, sep)
    __rdiv__ = __div__

m = MyString("abcd_abcd_abcd")
print m / "_"
print m / "_a"
print
print "_" / m
print "_a" / m
```

```
['abcd', 'abcd', 'abcd']
['abcd', 'bcd', 'bcd']

['abcd', 'abcd', 'abcd']
['abcd', 'bcd', 'bcd']
```

- __rdiv__ : 약속된 메소드
- __rdiv__ → __div__랑 같으나 객체가 오른쪽에 위치
- O + B를 B+ O로 쓰고 싶으면 __radd__ 사용
- 메소드 이름 앞에 'r'이 붙으면 연산자 오른쪽에 객체가 호출

▣ 연산자 중복

1. 수치 연산자 중복

```
class MyString:
    def __init__(self, str):
        self.str = str
    def __div__(self, sep):
        return str.split(self.str, sep)
    __rdiv__ = __div__
    def __neg__(self):
        t = list(self.str)
        t.reverse()
        return ''.join(t)
    __invert__ = __neg__

m = MyString("abcdef")
print -m
print ~m
```

```
fedcba
fedcba
```

▪~ 있는 것 → invert 연산

▣ 연산자 중복

2. 비교 연산자 중복

- 각각의 비교 연산에 대응되는 메소드 이름이 정해져 있지만 그러한 메소드가 별도로 정의되어 있지 않으면 cmp가 호출됨

메소드	연산자	비고
__cmp__(self, other)	아래 메소드가 부재한 상황에 호출되는 메소드	
__lt__(self, other)	self < other	
__le__(self, other)	self <= other	
__eq__(self, other)	self == other	
__ne__(self, other)	self != other	
__gt__(self, other)	self > other	
__ge__(self, other)	self >= other	

▪객체, 연산자(<), other 쓰면 lt 실행됨
▪lt → less than의 약자
▪메소드 le → less than or equal의 약자
▪== → 동등 연산 → equal의 약자
▪!= → not equal → ne 사용
▪> → greater than 의 약자 → gt
▪>= → ge → greater than or equal
▪__cmp__는 아래 메소드가 부재한 상황에서 호출되는 메소드

▣ 연산자 중복

2. 비교 연산자 중복

- 객체 c에 대한 c > 1연산의 행동 방식
  - c.__gt__()가 있다면 호출 결과을 그대로 반환
  - 정의된 c.__gt__()가 없고, __cmp__() 함수가 있을 경우
    * c.__cmp__() 호출 결과가 양수이면 True 반환, 아니면 False 반환

```
class MyCmp:
    def __cmp__(self, y):
        return 1 - y

c = MyCmp()
print c > 1 # c.__cmp__(1)을 호출, 반환값이 양수이어야 True
print c < 1 # c.__cmp__(1)을 호출, 반환값이 음수이어야 True
print c == 1 # c.__cmp__(1)을 호출, 반환값이 0이어야 True
```

```
False
False
True
```

▪1이 __cmp(self, y)에서 y에 들어감
▪c > 1 → cmp가 돌려주는 값이 양수여야 true 됨
▪c < 1 → cmp가 돌려주는 값이 음수여야 true 됨
▪c == 1 → cmp가 돌려주는 값이 0이여야 true 됨

▣ 연산자 중복

2. 비교 연산자 중복

- 객체 m에 대한 m < 10연산의 행동 방식
  - m.__lt__()가 있다면 호출 결과을 그대로 반환
  - 정의된 m.__lt__()가 없고, __cmp__() 함수가 있을 경우
    * m.__cmp__() 호출 결과가 음수이면 True 반환, 아니면 False 반환

```
class MyCmp2:
    def __lt__(self, y):
        return 1 < y

m = MyCmp2()
print m < 10 # m.__lt__(10)을 호출
print m < 2
print m < 1
```

```
True
True
False
```

▣ 연산자 중복

2. 비교 연산자 중복

- 객체 m에 대한 m == 10연산의 행동 방식
  - m.__eq__()가 있다면 호출 결과을 그대로 반환
  - 정의된 m.__eq__()가 없고, __cmp__() 함수가 있을 경우
    * m.__cmp__() 호출 결과가 0이면 True 반환, 아니면 False 반환

```
class MyCmp3:
    def __eq__(self, y):
        return 1 == y

m = MyCmp3()
print m == 10 # m.__eq__(10)을 호출
m1 = MyCmp3()
print m == 1

class MyCmp4:
    def __init__(self, value):
        self.value = value
    def __cmp__(self, other):
        if self.value == other:
            return 0
m2 = MyCmp4(10)
print m2 == 10
```

```
False
True
True
```

▪value = 10