파이썬 프로그래밍

클래스와 연산자 중복 정의

한국기술교육대학교
온라인평생교육원

▣ 문자열 변환과 호출 가능 객체

1. 문자열로 변환하기

1) __repr__
- 객체를 대표하여 유일하게 표현할 수 있는 공식적인 문자열
- eval() 함수에 의하여 같은 객체로 재생성 될 수 있는 문자열 표현

2) __str__
- 객체의 비공식적인 문자열 표현
- 사용자가 보기 편한 형태로 자유롭게 표현될 수 있음

```
class StringRepr:
    def __repr__(self):
        return 'repr called'
    def __str__(self):
        return 'str called'

s = StringRepr()
print s
print str(s)
print repr(s)
print `s`
```

```
str called
str called
repr called
repr called
```

▪__repr__에 대응되는 역함수가 존재 → eval() 함수
▪단순하게 s 프린트하면 __str__이 호출
▪repr(s) 내장함수 활용 시 __repr__이 호출
▪`s`로 프린트하면 __repr__이 호출
▪일반적으로 __str__이 가장 많이 사용됨

▣ 문자열 변환과 호출 가능 객체

1. 문자열로 변환하기

- __str__() 호출시
  - __str__()가 정의되어 있지 않으면 __repr__()이 대신 호출됨

```
class StringRepr:
    def __repr__(self):
        return 'repr called'

s = StringRepr()
print s
print repr(s)
print str(s)
print `s`
```

```
repr called
repr called
repr called
repr called
```

▪s와 str함수를 프린트할 때, __str__ 정의가 없으면 __repr__활용

▣ 문자열 변환과 호출 가능 객체

1. 문자열로 변환하기

- __repr__() 호출시
- __repr__()이 정의되어 있지 않으면 객체 식별자가 출력됨
- 대신하여 __str__()이 호출되지 않음

```
class StringRepr:
    def __str__(self):
        return 'str called'

s = StringRepr()
print s
print repr(s)
print str(s)
print `s`
```

```
str called
<__main__.StringRepr instance at 0x101d3f908>
str called
<__main__.StringRepr instance at 0x101d3f908>
```

▪repr()함수는 __str__을 찾지 않음

▪__repr__ 없으면 그대로 디폴트 객체 표현 양식이 출력됨

▣ 문자열 변환과 호출 가능 객체

2. 호출 가능한 클래스 인스턴스 만들기

- 클래스 인스턴스에 __call__ 메소드가 구현되어 있다면 해당 인스턴스는 함수와 같이 호출될 수 있다.

 - 인스턴스 x에 대해 x(a1, a2, a3)와 같이 호출된다면 x.__call(a1, a2, a3)__ 가 호출된다.

```
class Accumulator:
    def __init__(self):
        self.sum = 0
    def __call__(self, *args):
        self.sum += sum(args)
        return self.sum

acc = Accumulator()
print acc(1,2,3,4,5)
print acc(6)
print acc(7,8,9)
print acc.sum
```

```
15
21
45
45
```

▪함수는 호출가능한 객체

▪새로 만든 객체를 호출 가능한 객체로 만들려면 __call__사용

▪acc 인스턴스를 호출함

▪*args → 가변 인수 (여러 개의 인수를 튜플로 받아낼 수 있음)

▪self.sum에는 누적이 되어 있는 상태

▪acc라는 객체에 가로를 사용하여 호출이 가능하다고 명명함

▣ 문자열 변환과 호출 가능 객체

2. 호출 가능한 클래스 인스턴스 만들기

- 호출 가능 객체인지 알아보기

```
def check(func):
    if callable(func):
        print 'callable'
    else:
        print 'not callable'

class B:
    def func(self, v):
        return v
class A:
    def __call__(self, v):
        return v

a = A()
b = B()
check(a)
check(b)
print
print callable(a)
print callable(b)
```

```
callable
not callable

True
False
```

▣ 문자열 변환과 호출 가능 객체

2. 호출 가능한 클래스 인스턴스 만들기

- class A는 __call__이 존재 → a() 사용하면 __call__이 호출됨
- class B는 __call__이 없으므로 b() 형태로 사용 X
- check 내장함수 없이 바로 callable() 함수 사용 가능
- callable()함수의 인자로 class 이름 사용 가능
- class A, B 모두 callable 함 → 모든 클래스는 callable 함