33장

원하는 대로 클래스 바꾸기

33장 원하는 대로 클래스 바꾸기

33.1 특수 메서드 오버라이드하기

33.1 특수 메서드 오버라이드하기

특수 메서드

- 특수 메서드는 모두 밑줄 두 개로 시작해 밑줄 두 개로 끝난다.
- 여기 있는 메서드 이름 규칙은 파이썬에만 해당하며, 언어마다 특수 메서드 이름 규칙은 다르다.

구분	연산자	메서드 이름
수학 연산	+	_ _add_ _
	–	_ _sub_ _
	*	_ _mul_ _
	/	_ _truediv_ _
비교	==	_ _eq_ _
	〈	_ _lt_ _
	〉	_ _gt_ _
기타	print(),srt()	_ _str_ _
	객체 생성 (예: 변수 이름= 클래스 이름())	_ _init_ _

정의하는 클래스에 연산자를 사용할 수 있는 연산을 추가하려면 이런 특수 메서드를 오버라이드(override) 해야 한다.

- 오버라이드란 용어는 더 일반적인 파이썬 객체가 제공하는 기본적인 메서드 대신, 여러분이 정의하는 클래스 안에서 그 메서드를 다시 정의하면서 어떤 일을 수행할지 결정하는 것을 뜻한다.

33.1 특수 메서드 오버라이드하기

- 분수를 표현하는 새로운 객체 타입을 만드는 것부터 시작해 보자.
- 분수는 분자와 분모로 이뤄진다. 따라서 Fraction 객체의 데이터 속성은 정수 두 개다. 다음 코드는 Fraction 클래스의 기본 정의다.

```
class Fraction(object):
    def __init__(self, top, bottom):
        self.top = top
        self.bottom = bottom
```

---- 초기화 메서드는 매개변수를 두 개 받음

---- 매개변수를 가지고 데이터 속성을 초기화

코드 33-1
Fraction 클래스 정의

- 이 정의를 사용하면 다음과 같이 두 Fraction 객체를 만들고 더할 수 있다.

```
half = Fraction(1,2)
quarter = Fraction(1,4)
print(half + quarter)
```

- 1/2과 1/4을 더하면 3/4가 나온다. 하지만 프로그램을 실행하면 다음과 같은 오류가 표시된다.

```
TypeError: unsupported operand type(s) for +: 'Fraction' and 'Fraction'
```

- 이 메시지를 보니 파이썬은 두 Fraction 객체를 더하는 방법을 모른다. Fraction 객체 타입 안에 덧셈 연산을 정의하지 않았기 때문이다.

33.1 특수 메서드 오버라이드하기

- 덧셈은 두 객체 사이에 작동한다. 하나는 여러분이 메서드를 호출하는 대상 객체(self)이고, 다른 하나는 _ _add_ _ 메서드의 (명시적인)인자로 넘기는 Fraction 객체다.

```
class Fraction(object):
    def __init__(self, top, bottom):
        self.top = top
        self.bottom = bottom

    def __add__(self, other_fraction):
        new_top = self.top*other_fraction.bottom + \
                  self.bottom*other_fraction.top
        new_bottom = self.bottom*other_fraction.bottom
        return Fraction(new_top, new_bottom)

    def __mul__(self, other_fraction):
        new_top = self.top*other_fraction.top
        new_bottom = self.bottom*other_fraction.bottom
        return Fraction(new_top, new_bottom)
```

---- 두 Fraction 객체 사이의 + 연산자를 정의하기 위한 특수 메서드

---- 한 줄이 너무 길어서 백슬래시(\)를 사용해 두 줄로 나눔

덧셈의 분자 계산 ----

---- 덧셈의 분모 계산

---- 새로운 분모와 분자를 사용해 새 Fraction 객체를 만들어 반환

---- 두 Fraction 객체를 곱하는 메서드

코드 33-2
두 Fraction 객체를 더하는 메서드와 곱하는 메서드

33.1 특수 메서드 오버라이드하기

셀프 체크 33.1

33.2 str( ) 오버라이드하여 print( )가 객체를 출력하는 방법 변경

33.2 str( ) 오버라이드하여 print( )가 객체를 출력하는 방법 변경

» 두 Fraction 객체 사이의 + 연산자를 정의했으므로, 이전에 오류가 났던 코드를 제대로 사용할 수 있다.

```
half = Fraction(1,2)
quarter = Fraction(1,4)
print(half + quarter)
```

» 이 코드는 더 이상 오류가 나지는 않는다. 하지만 print()가 출력하는 값은 여전히 객체의 타입과 메모리 상의 위치다.

```
<__main__.Fraction object at 0x00000200B8BDC240>
```

33.2 str( ) 오버라이드하여 print( )가 객체를 출력하는 방법 변경

코드 33-3
Fraction 객체를 제대로 출력하기 위한 메서드

```
class Fraction(object):
    def __init__(self, top, bottom):
        self.top = top
        self.bottom = bottom

    def __add__(self, other_fraction):
        new_top = self.top*other_fraction.bottom + \
                  self.bottom*other_fraction.top
        new_bottom = self.bottom*other_fraction.bottom
        return Fraction(new_top, new_bottom)

    def __mul__(self, other_fraction):
        new_top = self.top*other_fraction.top
        new_bottom = self.bottom*other_fraction.bottom
        return Fraction(new_top, new_bottom)

    def __str__(self): ---- Fraction 객체를 출력하기 위해 문자열로 변환할 때 사용하는 메서드를 정의
        return str(self.top)+"/"+str(self.bottom) ---- 어떤 내용을 출력할지 지정하는 문자열을 반환
```

33.2 str( ) 오버라이드하여 print( )가 객체를 출력하는 방법 변경

» 이제 Fraction 객체에 대해 print를 사용하거나, 여러분이 만든 객체를 문자열로 바꾸기 위해str()를 사용하면 _ _str_ _이라고 정의한 특수 메서드를 호출한다.

- 예를 들어 다음 코드는 메모리 위치 대신 1/2를 출력한다.

```
half = Fraction(1, 2)
print(half)
```

- 다음 코드는 값이 1/2인 문자열 객체를 만든다.

```
half = Fraction(1, 2)
half_string = str(half)
```

» 셀프 체크 33.2

33.3 내부적으로 벌어지는 일들

33.3 내부적으로 벌어지는 일들

연산자를 사용하면 정확히 어떤 일이 벌어질까?

```
half = Fraction(1,2)
quarter = Fraction(1,4)
```

```
half + quarter
```

=

```
half.__add__(quarter)
```

모든 메서드 호출은 클래스 이름을 명시하고 self 매개변수에 객체를 명시하는 형식으로 바꿔 쓸 수 있다. 즉, 위 코드는 다음 코드와 같다

```
Fraction.__add__(half, quarter)
```

33.4 클래스로 할 수 있는 일은 무엇인가?

33.4.1 이벤트 스케줄링

» 예를 들어 영화제에 참가하면서 원하는 영화를 어떤 순서로 언제 볼지 정하고 싶은 경우다.

33.4.1.1 클래스를 쓰지 않는 경우

- 관람하려는 영화를 리스트에 모두 저장할 수 있다. 리스트의 각 원소는 보고 싶은 영화들이다.
- 영화와 관련된 정보로는 영화 제목, 시작 시간, 끝 시간이 있고, 원한다면 비평가가 매긴 평점을 넣을 수도 있다. 이런 정보를 튜플에 넣고, 튜플을 리스트의 원소로 만들수 있다.
- 하지만 이렇게 만든 리스트는 금방 다루기 곤란해진다. 예를 들어 모든 영화의 평점을 보고 싶다면 인덱싱을 두 번(한 번은 리스트를 인덱싱해서 원하는 영화를 표현하는 튜플을 찾고, 그렇게 찾은 튜플에서 다시 평점에 해당하는 값을 인덱싱해 가져와야 함) 해야 한다.
- 평점의 인덱스번호를 항상 기억해야 하고, 튜플에 새로운 필드를 추가하면 평점 위치가 달라질 수 있어서 튜플안에서 필드 위치를 함부로 바꾸기 어렵다는 단점도 있다.

33.4.1.2 클래스를 사용하는 경우

- Time 클래스는 시간을 표현하는 객체 타입이다. 이 객체 타입의 데이터 속성에는 시간(int),분(int), 초(int)가 있다. 이 클래스에 속한 객체에 대한 연산으로는 두 시간 사이의 차이를 계산하는 것과 전체 시간, 분, 초를 구하는 연산이 있다.
- Movie 클래스는 영화를 표현하는 객체 타입이다. 이 객체 타입의 데이터 속성으로는 이름(string), 시작 시간(Time), 끝 시간(Time), 평점(int)이 있다. 이 클래스에 속한 객체에 대한연산으로는 두 영화의 시간이 일부분이라도 겹치는지 알아보는 연산이나 영화 평점이 높은지 알아보는 연산 등이 있다.

33.5 요약

33.5 요약

- 특수 메서드는 미리 정해진 이름이 있으며, 이름의 앞과 뒤에 밑줄이 두 개씩(_ _) 들어간다.언어에 따라 다른 특수 메서드가 있을 수 있다.
- 특수 메서드를 사용하면 연산자를 활용하는 더 짧은 표현을 사용할 수 있다.

33.5 요약

» (Q33.1) 특수 메서드를 작성하여 Circle과 Stack에 대해 print() 함수를 사용해 보라. Stack 타입의 객체를 print하면 32장의 prettyprint와 똑같이 스택 내부의 객체들을 표시해야 한다.Circle 타입인 객체를 print하면 "circle: 1"이 표시돼야 한다(반지름이 1이 아니면 물론 다른 값이 표시돼야 한다). 이를 위해 Stack과 Circle의 _ _str_ _ 메서드를 구현해야 한다. 예를 들어 다음 코드는

```
circles = Stack()
one_circle = Circle()
one_circle.change_radius(1)
circles.add_one(one_circle)
two_circle = Circle()
two_circle.change_radius(2)
circles.add_one(two_circle)
print(circles)
```

다음과 같은 결과를 출력해야 한다.

```
|_ circle: 2 _|
|_ circle: 1 _|
```