

파이썬으로 배우는 데이터 구조



한동대학교
전산전자공학부
김영섭 교수



학습 목표

repr, str메소드들을 재정의(override)하여
객체를 문자열로 출력할 수 있다

Data Structures in Python

Chapter 1 - 2

- Object-Oriented Programming
- OOP in Python
- **OOP - Fraction Example**
- OOP - Classes
- OOP - In-Place Operators
- Exceptions
- Exception Clauses

Agenda

- Topics:
 - Model of objects in memory
 - Constructor
 - Using the Fraction class
 - Overriding default behavior
 - `__repr__`
 - `__str__`
- References:
 - [Problem Solving with Algorithms and Data Structures using Python](#)
 - Chapter 1.13 Object-Oriented Programming in Python

Example: Fractions

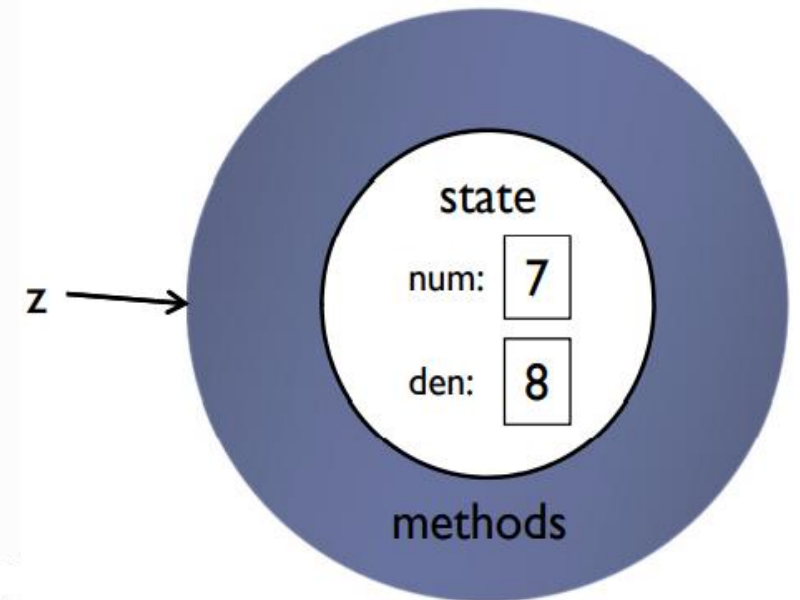
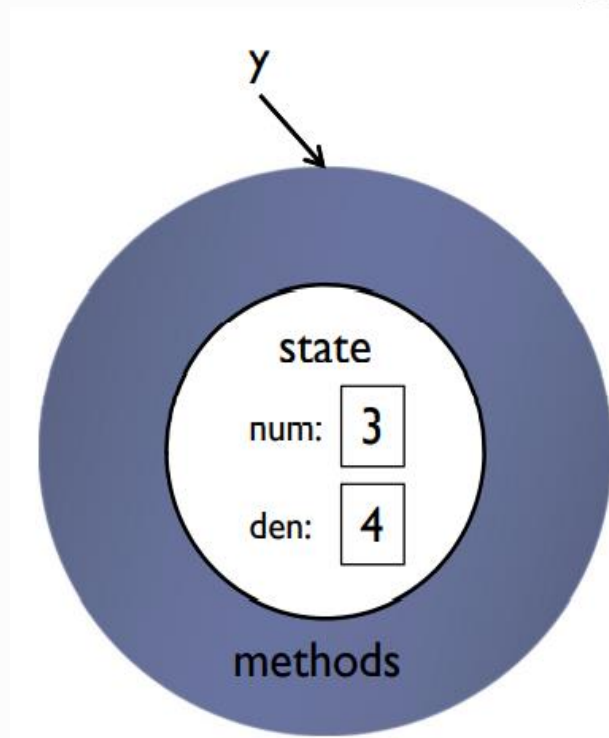
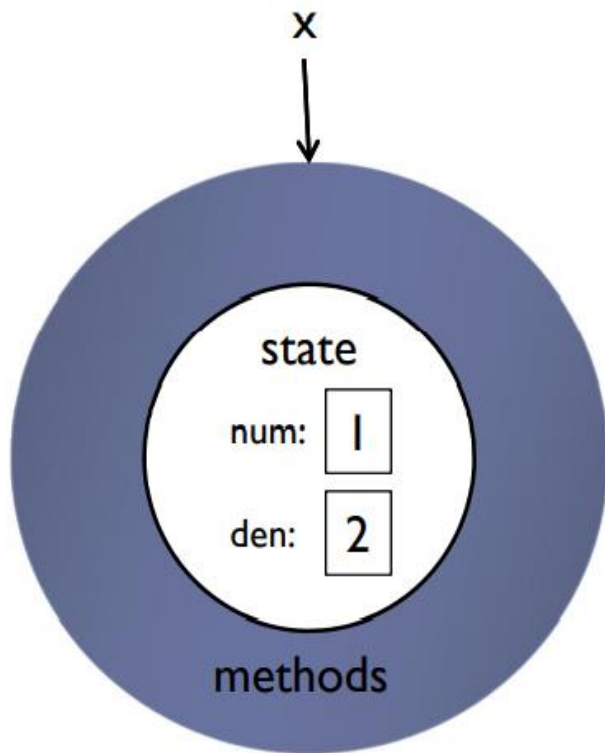
- Write a class to represent fractions in Python
 - create a fraction
 - add
 - subtract
 - multiply
 - divide
 - text representation



A diagram illustrating the components of a fraction. The fraction $\frac{1}{2}$ is shown in a large, bold font. An arrow points from the word "numerator" to the number "1". Another arrow points from the word "denominator" to the number "2".

Model of objects in memory

```
x = Fraction(1,2)  
y = Fraction(3,4)  
z = Fraction(7,8)
```



Constructor

- All classes must have a constructor
 - The constructor for a Fraction should store the numerator and the denominator

```
class Fraction:  
    def __init__(self, top, bottom):  
        self.num = top          #numerator  
        self.den = bottom      #denominator
```

Using the Fraction class

- So far, we can create a Fraction object:

```
x = Fraction(3, 4)
```

- We can access the state variables directly
 - Although **not generally good** practice to do so

x.num	3
x.den	4

- What else can we do with Fractions?
 - Nothing yet. We need to write the functions first!

Overriding default behavior

- All classes get a number of methods provided by default
 - Since default behavior is **not** very useful, we should write our own versions of those methods. It is called "override" (재정의하다).
 - `__repr__`
 - `__str__`

x.num	3
x.den	4
print(x)	3/4



```
[4]: x
[4]: Fraction(3,4)
```



```
[5]: print(x)
      3/4
```

```
[1]: class Fraction:
      def __init__(self, top, bottom):
          self.num = top          #numerator
          self.den = bottom       #denominator

[3]: x = Fraction(3, 4)
      x.num

[3]: 3

[4]: x.den

[4]: 4

[5]: x                                Without the __repr__ method
[5]: <__main__.Fraction at 0x1ef99f657c0>

[6]: print(x)                         Without the __str__ method
[6]: <__main__.Fraction object at 0x000001EF99F657C0>
```

Aside: Use of string formatting syntax

- Often we want to use a string that combines literal text and information from variables

```
name = 'Andrew'  
greeting = 'Hello ' + name + '. How are you?'
```

- We can use string formatting to perform this task
 - Use curly braces within the string to signify a variable to be replaced

```
host = 'Andrew'  
greeting = 'Hello {one}. How are you?'.format(one=host)
```

- We can put the argument position in the curly braces

```
host = 'Park'  
guest = 'Lee'  
greeting = 'Hello {one} {two}'.format(two=guest, one=host)
```

```
greeting = 'Hello {0} {1}'.format(host, guest)
```

```
greeting = f'Hello {host} {guest}'
```

`__repr__`

- The `__repr__()` produces a string that unambiguously describes the object.
 - All classes should have a `__repr__` function implemented.
 - Ideally, the representation could be used to create the object

```
class Fraction:
    def __init__(self, top, bottom):
        self.num = top
        self.den = bottom
    def __repr__(self):
        return 'Fraction({}, {})'.format(self.num, self.den)
```

- With `__repr__()`, we can use the print function to print the object
 - Using `__repr__()`, but not `__str__()`

```
x = Fraction(2,3)
x
```

Fraction(2,3)

```
x = Fraction(2,3)
print(x)
```

Fraction(2,3)

`__str__`

- The `__str__()` method returns a string representing the object
 - By default, it calls the `__repr__` method
 - The `__str__` method should focus on being human readable
 - We should implement a version with a natural representation:

```
def __str__(self):  
    return str(self.num) + '/' + str(self.den)
```

- With `__str__()`, we can use the print function to print the object
 - Using `__repr__()`, and `__str__()`

```
x = Fraction(2,3)
```

```
x
```

```
Fraction(2,3)
```

```
x = Fraction(2,3)
```

```
print(x)
```

```
2/3
```

__str__ and __repr__

- What is the difference between the __str__ and __repr__ methods of a Python object?
 - In short __repr__ goal is to be unambiguous and __str__ is to be readable.
 - The official Python documentation says:
 - __repr__ is used to compute the “official” string representation of an object.
 - __str__ is used to compute the “informal” string representation of an object.
 - The print statement and str() built-in function uses __str__
 - The repr() built-in function uses __repr__ to display the object.

```
s1 = Square(10)  
str(s1)
```

'10 x 10 Square'

```
repr(s1)
```

'Square(10)'

- Resource:
 - <https://stackoverflow.com/questions/1436703/what-is-the-difference-between-str-and-repr>

Exercise 1

1. Write the `__str__` and `__repr__` method for the Square class in Geometry.py.
2. Why would it be useful to implement a `__str__` method?

```
s = Square(10)
print(s)
print(str(s))
print(s.__repr__())
print(repr(s))
```

```
10 x 10 Square
10 x 10 Square
Square(10)
Square(10)
```

Exercise 2

- Consider the Circle class which we developed previously:
 - Modify the constructor with default values of 0 for the radius
 - Write the `__str__` method and the `__repr__` method
 - Sample Run:

```
c1 = Circle(10)
```

```
str(s1)      'A circle with a radius of 10cm'
```

```
repr(s1)     'Circle(10)'
```

학습 정리

- 1) repr() 함수는 개발자가 해당 객체를 생성할 때 필요한 문자열을 반환한다
- 2) str() 함수는 객체의 내용을 사용자가 이해하기 쉬운 문자열로 반환한다

파이썬으로 배우는 데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다