

파이썬으로 배우는 데이터 구조



한동대학교
전산전자공학부
김영섭 교수



학습 목표

분수(Fraction) 클래스에서 필요한
연산자 중복 정의(overloading)를 할 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

하나님은 모든 사람이 구원을 받으며 진리를 아는데 이르기를 원하시느니라 (딤후2:4)

내 아들들을 먼 곳에서 이끌며 내 딸들을 땅 끝에서 오게 하며 내 이름으로 불려지는 모든 자 곧 내가 내 영광을 위하여 창조한 자를 오게 하라 그를 내가 지었고 그를 내가 만들었노라 (사43:6-7)

너는 청년의 때에 너의 창조주를 기억하라 곧 곤고한 날이 이르기 전에, 나는 아무 낙이 없다고 할 해들이 가깝기 전에 (전12:1)

그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전10:31)

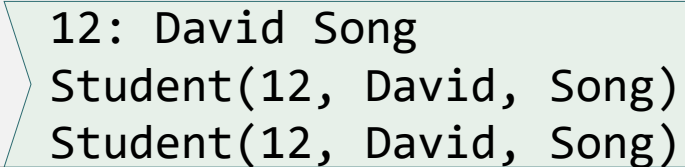
Agenda

- **Classes**
 - Overloading Operators
 - `__add__`, `__sub__`, `__eq__`
 - GCD
 - `__lt__`
- In-Place Operations
 - `__mul__`, `__rmul__`, `__imul__`
- References:
 - [Problem Solving with Algorithms and Data Structures using Python](#)
 - Chapter 1.13 Object-Oriented Programming in Python
 - [Chapter 2.2 A Proper Class](#)

Exercise

- Create a **Student** class:
 - The Student class should have three attributes: id, last_name, and first_name.
 - Create a constructor to initialize the values
 - Implement the `__repr__`, `__str__` and `__eq__` methods
- Sample Run:

```
s1 = Student(12, 'David', 'Song')
print(s1)
print(s1.__repr__())
s1
```



12: David Song
Student(12, David, Song)
Student(12, David, Song)

Reminder: Fraction class

- 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. An arrow points from the word "numerator" to the number 1. Another arrow points from the word "denominator" to the number 2.

Overloading Operators

- Python operators work for built-in classes.
 - But same operator behaves differently with different types.
 - E.g. the + operator:
 - performs arithmetic addition on two numbers,
 - merges two lists,
 - concatenates two strings.
 - Allow same operator to have different meaning according to the context is called **operator overloading**(연산자 오버딩).

Operator	Expression	Internally
Addition	<code>f1 + f2</code>	<code>f1.__add__(f2)</code>
Subtraction	<code>f1 - f2</code>	<code>f1.__sub__(f2)</code>
Equality	<code>f1 == f2</code>	<code>f1.__eq__(f2)</code>

`__add__`

- The `__add__` method is called when the `+` operator is used.
 - If we implement `__add__` then we can use `+` to add the objects.
 - `f1 + f2` gets translated into `f1.__add__(f2)`
 - Sample Run:

```
x = Fraction(1, 2)
y = Fraction(1, 4)
z = x + y
print(z)
```

6/8

```
= self + other
= 1/2 + 1/4
= (1 * 4 + 1 * 2) / (2 * 4)
= 6/8
```

- Code:

```
def __add__(self, other):
```

__add__

- The `__add__` method is called when the `+` operator is used.
 - If we implement `__add__` then we can use `+` to add the objects.
 - `f1 + f2` gets translated into `f1.__add__(f2)`
 - Sample Run:

```
x = Fraction(1, 2)
y = Fraction(1, 4)
z = x + y
print(z)
```

6/8

```
= self + other
= 1/2 + 1/4
= (1 * 4 + 1 * 2) / (2 * 4)
= 6/8
```

- Code:

```
def __add__(self, other):
    num = self.num * other.den + self.den * other.num
    den = self.den * other.den
    return Fraction(num, den)
```

__sub__

- The `__sub__` method is called when the `-` operator is used.
 - If we implement `__sub__` then we can use `-` to do subtraction.
 - `f1 - f2` gets translated into `f1.__sub__(f2)`
 - Sample Run:

```
x = Fraction(1, 2)
y = Fraction(1, 4)
z = x - y
print(z)
```

2/8

```
= self - other
= 1/2 - 1/4
= (1 * 4 - 1 * 2) / (2 * 4)
= 2/8
```

- Code:

```
def __sub__(self, other):
```

`__eq__`

- The `__eq__` method checks equality of the objects.
 - Default behavior is to compare the references.
 - We want to compare the contents.
 - Sample Run:

```
x = Fraction(12, 30)
y = Fraction(2, 5)
print(x == y)
```

True

```
x = Fraction(4, 1)
y = Fraction(1, 4)
print(x == y)
```

False

- Code:

```
def __eq__(self, other):
```

`__eq__`

- The `__eq__` method checks equality of the objects.
 - Default behavior is to compare the references.
 - We want to compare the contents.
 - Sample Run:

```
x = Fraction(12, 30)
y = Fraction(2, 5)
print(x == y)
```

True

```
x = Fraction(4, 1)
y = Fraction(1, 4)
print(x == y)
```

False

- Code:

```
def __eq__(self, other):
```

```
= (self == other)
= (12/30 == 2/5)
= (12 * 5 == 2 * 30)
= (60 == 60)
```

Exercise 1

- What is the output of the following code?

```
x = Fraction(2, 3)
y = Fraction(1, 3)
z = y + y
print(x)
print(z)
print(x == z)
```

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

Exercise 1

- What is the output of the following code?

```
x = Fraction(2, 3)
y = Fraction(1, 3)
z = y + y
print(x)
print(z)
print(x == z)
```

```
2/3
1/3
True
```

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

```
AttributeError: 'int' object
has no attribute 'den'
```

Improving `__eq__`

- Check the type of the other operand.
 - If the type is not a Fraction, then not equal?
 - What other decisions could we make for equality?

```
def __eq__ (self, other):  
    if not isinstance(other, Fraction):  
        return False  
    return self.num * other.den == other.num * self.den
```

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

False

Improving your code

- Fractions:
 - $12/30$
 - $2/5$
- The first fraction can be simplified to $2/5$.
- The Common Factors of 12 and 30 were 1, 2, 3 and 6.
- The Greatest Common Factor is 6.
 - So the largest number we can divide both 12 and 30 evenly by is 6.
- And so **$12/30$** can be simplified to **$2/5$** .

Greatest Common Divisor

- Use Euclid's Algorithm.
 - Given two numbers, n and m , find the number k , such that k is the largest number that evenly divides both n and m .
 - Example: Find the GCD of 270 and 192,
 - $\text{gcd}(270, 192)$: $m=270, n=192$ ($m \neq 0, n \neq 0$)
 - Use long division to find that $270/192 = 1$ with a remainder of 78.
We can write this as: $\text{gcd}(270, 192) = \text{gcd}(192, 78)$
 - $\text{gcd}(192, 78)$: $m=192, n=78$ ($m \neq 0, n \neq 0$)
 - $192/78 = 2$ with a remainder of 36.
We can write this as: $\text{gcd}(192, 78) = \text{gcd}(78, 36)$
 - $\text{gcd}(78, 36)$: $m=78, n=36$ ($m \neq 0, n \neq 0$)
 - $78/36 = 2$ with a remainder of 6.
 - $\text{gcd}(78, 36) = \text{gcd}(36, 6)$
 - $\text{gcd}(36, 6)$: $m=36, n=6$ ($m \neq 0, n \neq 0$)
 - $36/6 = 6$ with a remainder of 0
 - $\text{gcd}(36, 6) = \text{gcd}(6, 0) = 6$

```
def gcd(m, n):  
    while m % n != 0:  
        old_m = m  
        old_n = n  
        m = old_n  
        n = old_m % old_n  
    return n
```

Improve the constructor

- We can improve the constructor so that it always represents a fraction using the "lowest terms" form.
 - What other things might we want to add to a Fraction?

```
class Fraction:
    def __init__(self, top, bottom):
        gcd = Fraction.gcd(top, bottom) # get gcd
        self.num = top // gcd
        self.den = bottom // gcd

    def gcd(m, n):
        while m % n != 0:
            old_m, old_n = m, n
            m = old_n
            n = old_m % old_n
        return n
```

```
def gcd(m,n):
    while m % n != 0:
        m, n = n, m % n
    return n
```

Sample Run:

- **Without** the GCD

```
x = Fraction(12,30)
y = Fraction(2, 5)
print (x == y)
print(x)
print(y)
```

```
True
12/30
2/5
```

- **With** the GCD

```
x = Fraction(12,30)
y = Fraction(2, 5)
print (x == y)
print(x)
print(y)
```

```
True
2/5
2/5
```

Other standard Python operators

- Many standard operators and functions:

- <https://docs.python.org/3.9/library/operator.html>

- Common Arithmetic operators

- `object.__add__(self, other)`
 - `object.__sub__(self, other)`
 - `object.__mul__(self, other)`
 - `object.__truediv__(self, other)`

- Common Relational operators

- `object.__lt__(self, other)`
 - `object.__le__(self, other)`
 - `object.__eq__(self, other)`
 - `object.__ne__(self, other)`
 - `object.__gt__(self, other)`
 - `object.__ge__(self, other)`

- **In-place** arithmetic operators

- `object.__iadd__(self, other)`
 - `object.__isub__(self, other)`
 - `object.__imul__(self, other)`
 - `object.__itruediv__(self, other)`

`+=`
`-=`
`*=`
`/=`

- Reversed versions

- `object.__radd__(self, other)`
 - `object.__rsub__(self, other)`
 - `object.__rmul__(self, other)`
 - `object.__rdiv__(self, other)`
 - ...

Exercise 2

- Implement the `__truediv__` of the Fraction class:
- Sample Run:

```
a = Fraction(1, 3)
b = Fraction(4, 5)
d = a / b
print (d)
```

5/12

- Code

```
def __truediv__(self, other):
```

```
= (self / other)
= (1/3 / 4/5)
= (1 * 5 / 3 * 4)
= (5 / 12)
```

Exercise 2 solution

Solution

- Implement the `__truediv__` of the Fraction class:
- Sample Run:

```
a = Fraction(1, 3)
b = Fraction(4, 5)
d = a / b
print (d)
```

5/12

- Code:

```
def __truediv__(self, other):
    num = self.num * other.den
    den = self.den * other.num
    return Fraction(num, den)
```

```
= (self / other)
= (1/3 / 4/5)
= (1 * 5 / 3 * 4)
= (5 / 12)
```

Exercise 3

- Implement the `__lt__` method to compare two Fraction objects:
- Sample Run:

```
a = Fraction(1, 3)
b = Fraction(4, 5)
if a < b:
    print("a < b")
else:
    print("a >= b")
```

a < b

- Code

```
def __lt__(self, other):
```

```
= (self < other)
= (1/3 / 4/5)
= 5, 12
= 5 < 12
```


Summary

- A class is a template, a blueprint and a data type for objects.
- A class defines the **data fields** of objects and provides an initializer for initializing objects and other **methods** for manipulating the data.
- The initializer always named **`__init__`**.
The first parameter in each method including the initializer in the class refers to the object that calls the methods, i.e., **`self`**.
- Data fields in classes should be **hidden** to prevent data tampering and to make class easy to maintain. - Encapsulation(은닉화)
- We can **override(재정의) the default methods** in a class definition.

학습 정리

- 1) self를 사용하여 더하기(add), 빼기(sub), 비교(eq) 등의 연산을 정의한다
- 2) 최대공약수(GCD) 함수를 이용해 분수를 간단한 형태로 나타낼 수 있다

파이썬으로 배우는 데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다