Object-Oriented Programming

Jerry Cain
CS 106AX
November 1, 2019
slides leveraged from those constructed by Eric Roberts

The Principles of OOP

- Object-oriented programming (often abbreviated to OOP) was invented in Norway in the 1960s but was not adopted widely for more than a decade.
- Object-oriented programming is defined by two principles, both of which I mentioned on Wednesday during my discussion of classes and objects in Python:
  - **Encapsulation**—The principle that data values and the methods that manipulate those values should be integrated into a single coherent structure called an object.
  - **Inheritance**—The idea that objects and the classes that those objects represent form hierarchies that allow new classes to share behavior with classes at higher levels in the hierarchy.
- Today’s lecture focuses on the use of encapsulation to define both the values and operations on rational numbers.

Rational Numbers

- Section 9.3 illustrates the idea of encapsulation by defining a class called `Rational` to represent rational numbers, which are simply the quotient of two integers.
- Rational numbers can be useful in cases in which you need exact calculation with fractions. Even if you use a `double`, the floating-point number 0.1 is represented internally as an approximation. The rational number 1 / 10 is exact.
- Rational numbers support the standard arithmetic operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>( \frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd} )</td>
</tr>
<tr>
<td>Subtraction</td>
<td>( \frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd} )</td>
</tr>
<tr>
<td>Multiplication</td>
<td>( \frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd} )</td>
</tr>
<tr>
<td>Division</td>
<td>( \frac{a}{b} \div \frac{c}{d} = \frac{ad}{bc} )</td>
</tr>
</tbody>
</table>

Implementing the Rational Class

- The next three slides show the initial version of the `Rational` class along with some brief annotations.
- As you read through the code, the following features are worth special attention:
  - **The constructor checks that rational numbers obey certain rules.** These rules are described in more detail in the text but include reducing the fraction to lowest terms.
  - **For new, operations are specified using the receiver syntax.** When you apply an operator to two `Rational` values, one of the operands is the receiver and the other is passed as an argument, as in `r1.add(r2)`.

The Rational Class

```python
# File: rational.py

---
This module defines a class for representing rational numbers.
---
import math
class Rational:
    # Implementation note
    # The Rational class ensures that every number has a unique
    # internal representation by guaranteeing that the following
    # conditions hold:
    # 1. The denominator must be greater than 0.
    # 2. The numerator is always represented as 0/1.
    # 3. The fraction is always reduced to lowest terms.

def __init__(self, num, den=1):
    """Create a new Rational object from num and den."""
    if den == 0:
        raise ValueError(’Illegal denominator value’)
    if num == 0:
        den = 1
    elif den < 0:
        den = -den
    num = ~num
    g = math.gcd(abs(num), den)
    self._num = num // g
    self._den = den // g

def __str__(self):
    """Return the string representation of this object."""
    if self._den == 1:
        return str(self._num)
    else:
        return str(self._num) + ' / ' + str(self._den)
```

The Rational Class
The Rational Class

```python
def add(self, rhs):
    """Creates a new Rational by adding r to self."""
    return Rational(self.num * rhs.den + self.den * rhs.num,
                    self.den * rhs.den)
def sub(self, rhs):
    """Creates a new Rational by subtracting r from self."""
    return Rational(self.num * rhs.den - self.den * rhs.num,
                    self.den * rhs.den)
def mul(self, rhs):
    """Creates a new Rational by multiplying self by r."""
    return Rational(self.num * rhs.num, self.den * rhs.den)
def div(self, rhs):
    """Creates a new Rational by dividing self by r."""
    return Rational(self.num * rhs.den, self.den * rhs.num)
```

Simulating Rational Calculation

- The next slide works through all the steps in the calculation of a simple program that adds three rational numbers.

\[ \frac{1}{2} + \frac{1}{3} + \frac{1}{6} \]

- With rational arithmetic, the computation is exact. If you use floating-point arithmetic, the result looks like this:

```python
PyCharm
>>> 1/2 + 1/3 + 1/6
0.9999999999999999
```
## Operator Methods in Python

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__add__</code></td>
<td>Redefines the <code>+</code> operator</td>
</tr>
<tr>
<td><code>__sub__</code></td>
<td>Redefines the <code>-</code> operator</td>
</tr>
<tr>
<td><code>__mul__</code></td>
<td>Redefines the <code>*</code> operator</td>
</tr>
<tr>
<td><code>__truediv__</code></td>
<td>Redefines the <code>/</code> operator</td>
</tr>
<tr>
<td><code>__floordiv__</code></td>
<td>Redefines the <code>//</code> operator</td>
</tr>
<tr>
<td><code>__mod__</code></td>
<td>Redefines the <code>%</code> operator</td>
</tr>
<tr>
<td><code>__pow__</code></td>
<td>Redefines the <code>**</code> operator</td>
</tr>
<tr>
<td><code>__neg__</code></td>
<td>(not applicable) Redefines the unary <code>-</code> operator</td>
</tr>
<tr>
<td><code>__eq__</code></td>
<td>(symmetric) Redefines the <code>==</code> operator</td>
</tr>
<tr>
<td><code>__ne__</code></td>
<td>(symmetric) Redefines the <code>!=</code> operator</td>
</tr>
<tr>
<td><code>__lt__</code></td>
<td>(inferred from <code>&gt;</code>) Redefines the <code>&lt;</code> operator</td>
</tr>
<tr>
<td><code>__gt__</code></td>
<td>(inferred from <code>&lt;</code>) Redefines the <code>&gt;</code> operator</td>
</tr>
<tr>
<td><code>__le__</code></td>
<td>(inferred from <code>&gt;=</code>) Redefines the <code>&lt;=</code> operator</td>
</tr>
<tr>
<td><code>__ge__</code></td>
<td>(inferred from <code>&lt;=</code>) Redefines the <code>&gt;=</code> operator</td>
</tr>
</tbody>
</table>

The End