Solutions — Object Oriented Programming

Jonathan K. Vis

Dept. of Human Genetics, Leiden University Medical Center

`j.k.vis@lumc.nl`

0. Skeleton

fraction.py

```
class Fraction(object):
    def __init__(self, numerator, denominator=1):
        self._numerator = numerator
        self._denominator = denominator
```

test_fraction.py

```
>>> f1 = Fraction(1, 2)
```

1. Adding a print representation

`fraction.py`

```python
class Fraction(object):
    def __init__(self, numerator, denominator=1):
        self._numerator = numerator
        self._denominator = denominator

    def __str__(self):
        return str(self._numerator) + '/' + str(self._denominator)
```

`test_fraction.py`

```python
>>> print(Fraction(1, 2))
1/2
```

2. Add the + operator

`fraction.py`

```python
class Fraction(object):
    def __init__(self, numerator, denominator=1):
        self._numerator = numerator
        self._denominator = denominator

    def __str__(self):
        return str(self._numerator) + '/' + str(self._denominator)

    def __add__(self, other):
        return Fraction(self._numerator * other._denominator +
                        other._numerator * self._denominator,
                        self._denominator * other._denominator)
```

`test_fraction.py`

```python
>>> print(Fraction(1, 2) + Fraction(2, 3))
7/6
```

3. Add an invert function

`fraction.py`

```python
class Fraction(object):
    ...

    def __str__(self):
        return str(self._numerator) + '/' + str(self._denominator)

    def __add__(self, other):
        return Fraction(self._numerator * other._denominator +
                        other._numerator * self._denominator,
                        self._denominator * other._denominator)

    def invert(self):
        return Fraction(self._denominator, self._numerator)
```

`test_fraction.py`

```python
>>> print(Fraction(1, 2).invert())
2/1
```

4. Conversion to a decimal representation

`fraction.py`

```python
class Fraction(object):
    ...

    def __add__(self, other):
        return Fraction(self._numerator * other._denominator +
                        other._numerator * self._denominator,
                        self._denominator * other._denominator)

    def invert(self):
        return Fraction(self._denominator, self._numerator)

    def to_float(self):
        return self._numerator / self._denominator
```

`test_fraction.py`

```python
>>> print(Fraction(1, 2).to_float())
0.5
```

5. Get the integer part of a fraction

`fraction.py`

```python
class Fraction(object):
    ...

    def invert(self):
        return Fraction(self._denominator, self._numerator)

    def to_float(self):
        return self._numerator / self._denominator

    def integer(self):
        return self._numerator // self._denominator
```

`test_fraction.py`

```python
>>> print(Fraction(7, 6).integer())
1
```

6a. Substracting

fraction.py

```python
class Fraction(object):
    ...

    def to_float(self):
        return self._numerator / self._denominator

    def integer(self):
        return self._numerator // self._denominator

    def __sub__(self, other):
        return Fraction(self._numerator * other._denominator -
                        other._numerator * self._denominator,
                        self._denominator * other._denominator)
```

test_fraction.py

```python
>>> print(Fraction(1, 2) - Fraction(2, 3))
-1/6
```

6b. Multiplication

`fraction.py`

```python
class Fraction(object):
    ...

    def integer(self):
        return self._numerator // self._denominator

    def __sub__(self, other):
        return Fraction(self._numerator * other._denominator -
                        other._numerator * self._denominator,
                        self._denominator * other._denominator)

    def __mul__(self, other):
        return Fraction(self._numerator * other._numerator,
                        self._denominator * other._denominator)
```

`test_fraction.py`

```python
>>> print(Fraction(1, 2) * Fraction(2, 3))
2/6
```

6c. Division

fraction.py

```python
class Fraction(object):
    ...

    def __sub__(self, other):
        return Fraction(self._numerator * other._denominator -
                        other._numerator * self._denominator,
                        self._denominator * other._denominator)

    def __mul__(self, other):
        return Fraction(self._numerator * other._numerator,
                        self._denominator * other._denominator)

    def __div__(self, other):
        return self * other.invert()
```

test_fraction.py

```python
>>> print(Fraction(1, 2) / Fraction(2, 3))
3/4
```

7. Simplification

fraction.py

```python
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)


class Fraction(object):
    ...

    def simplify(self):
        divisor = gcd(self._numerator, self._denominator)
        self._numerator = self._numerator // divisor
        self._denominator = self._denominator // divisor
        return self
```

test_fraction.py

```python
>>> print(Fraction(2, 6).simplify())
1/3
```

8–12. More advanced stuff

8. It would be a good idea to simplify() after each arithmetic operator.

9. Here, you really need the fractions to be in a *normal* form.

10. Probably (for printing reasons) you would like for the numerator to be negative and not the denominator, avoid 1/-4. And convert to positive when both the numerator and denominator are negative.

11. Something like:

```python
def __mul__(self, other):
    if isinstance(other, int):
        return Fraction(self._numerator * other, self._denominator)
    return Fraction(self._numerator * other._numerator,
                    self._denominator * other._denominator)
```

12. Using isinstance(numerator, int) etc. in the __init__ function.

General remarks

- As always, more docstrings and *more* comments;
- All imports on the top of the file;
- a is b does *not* check for numerical equality: use ==;
- *Never* use bitwise operators: &, |, ^, ~: use and or not;
- Prefer not calling the special functions, e.g., __add__ directly: use +;
- Try to make your code look *beautiful*.