fractions — Rational Numbers

Purpose: Implements a class for working with rational numbers.

The `Fraction` class implements numerical operations for rational numbers based on the API defined by `Rational` in the `numbers` module.

Creating Fraction Instances

As with the decimal module, new values can be created in several ways. One easy way is to create them from separate numerator and denominator values:

```
# fractions_create_integers.py

import fractions

for n, d in [(1, 2), (2, 4), (3, 6)]:
    f = fractions.Fraction(n, d)
    print('{}/{} = {}'.format(n, d, f))
```

The lowest common denominator is maintained as new values are computed.

```
$ python3 fractions_create_integers.py

1/2 = 1/2
2/4 = 1/2
3/6 = 1/2
```

Another way to create a `Fraction` is using a string representation of `<numerator> / <denominator>`:

```
# fractions_create_strings.py

import fractions

for s in ['1/2', '2/4', '3/6']:
    f = fractions.Fraction(s)
    print('{} = {}'.format(s, f))
```

The string is parsed to find the numerator and denominator values.

```
$ python3 fractions_create_strings.py

1/2 = 1/2
2/4 = 1/2
3/6 = 1/2
```

Strings can also use the more usual decimal or floating point notation of series of digits separated by a period. Any string that can be parsed by `float()` and that does not represent “not a number” (`NaN`) or an infinite value is supported.

```
# fractions_create_strings_floats.py

import fractions

for s in ['0.5', '1.5', '2.0', '5e-1']:
    f = fractions.Fraction(s)
    print('{0:>4} = {1}'.format(s, f))
```

The numerator and denominator values represented by the floating point value is computed automatically.

```
$ python3 fractions_create_strings_floats.py
```

```
 0.5 = 1/2
 1.5 = 3/2
 2.0 = 2
5e-1 = 1/2
```

It is also possible to create `Fraction` instances directly from other representations of rational values, such as `float` or `Decimal`.

```python
# fractions_from_float.py

import fractions

for v in [0.1, 0.5, 1.5, 2.0]:
    print('{} = {}'.format(v, fractions.Fraction(v)))
```

Floating point values that cannot be expressed exactly may yield unexpected results.

```
$ python3 fractions_from_float.py

0.1 = 3602879701896397/36028797018963968
0.5 = 1/2
1.5 = 3/2
2.0 = 2
```

Using `Decimal` representations of the values gives the expected results.

```python
# fractions_from_decimal.py

import decimal
import fractions

values = [
    decimal.Decimal('0.1'),
    decimal.Decimal('0.5'),
    decimal.Decimal('1.5'),
    decimal.Decimal('2.0'),
]

for v in values:
    print('{} = {}'.format(v, fractions.Fraction(v)))
```

The internal implementation of `Decimal` does not suffer from the precision errors of the standard floating point representation.

```
$ python3 fractions_from_decimal.py

0.1 = 1/10
0.5 = 1/2
1.5 = 3/2
2.0 = 2
```

Arithmetic

Once the fractions are instantiated, they can be used in mathematical expressions.

```python
# fractions_arithmetic.py

import fractions

f1 = fractions.Fraction(1, 2)
f2 = fractions.Fraction(3, 4)

print('{} + {} = {}'.format(f1, f2, f1 + f2))
print('{} - {} = {}'.format(f1, f2, f1 - f2))
print('{} * {} = {}'.format(f1, f2, f1 * f2))
print('{} / {} = {}'.format(f1, f2, f1 / f2))
```

All of the standard operators are supported.

```
$ python3 fractions_arithmetic.py

1/2 + 3/4 = 5/4
1/2 - 3/4 = -1/4
1/2 * 3/4 = 3/8
1/2 / 3/4 = 2/3
```

Approximating Values

A useful feature of `Fraction` is the ability to convert a floating point number to an approximate rational value.

```python
# fractions_limit_denominator.py

import fractions
import math

print('PI       =', math.pi)

f_pi = fractions.Fraction(str(math.pi))
print('No limit =', f_pi)

for i in [1, 6, 11, 60, 70, 90, 100]:
    limited = f_pi.limit_denominator(i)
    print('{0:8} = {1}'.format(i, limited))
```

The value of the fraction can be controlled by limiting the size of the denominator.

```
$ python3 fractions_limit_denominator.py

PI       = 3.141592653589793
No limit = 3141592653589793/1000000000000000
       1 = 3
       6 = 19/6
      11 = 22/7
      60 = 179/57
      70 = 201/64
      90 = 267/85
     100 = 311/99
```

See also

- Standard library documentation for fractions
- `decimal` – The `decimal` module provides an API for fixed and floating point math.
- `numbers` – Numeric abstract base classes.
- Python 2 to 3 porting notes for fractions

decimal — Fixed and Floating Point Math

random — Pseudorandom Number Generators

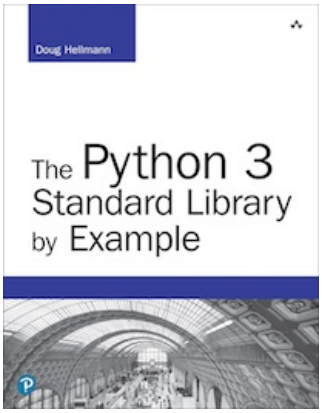
Quick Links

Creating Fraction Instances
Arithmetic
Approximating Values

This page was last updated 2016-12-28.

Navigation

decimal — Fixed and Floating Point Math
random — Pseudorandom Number Generators



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?