Operator Overloading

1. OOP to count Flops
   - a flop = a floating-point operation
   - overloading arithmetical operators

2. Quaternions
   - hypercomplex numbers
   - application in computer graphics
1. OOP to count Flops
   - a flop = a floating-point operation
   - overloading arithmetical operators

2. Quaternions
   - hypercomplex numbers
   - application in computer graphics
A flop is short for floating-point operation.

In scientific computation, the cost analysis is often measured in flops.

Note: before version 6, MATLAB had a `flops` command.

Using Object Oriented Programming:

1. we define a class `FlopFloat`,
2. every object stores its `#flops`: these are the flops used to compute the number,
3. the overloaded arithmetical operators count also the flops for each result.
Operator Overloading

1. OOP to count Flops
   - a flop = a floating-point operation
   - overloading arithmetical operators

2. Quaternions
   - hypercomplex numbers
   - application in computer graphics
overloading operators

Recall the addition of strings:

```python
>>> "ab" + "bc"
'abbc'
```

and the addition of lists:

```python
>>> [1,2] + [2,3]
[1, 2, 2, 3]
```

The `+` operator is defined via the `__add__` method:

```python
>>> L = [1,2]
>>> L.__add__([3,4])
[1, 2, 3, 4]
```
comparison operators and methods

<table>
<thead>
<tr>
<th>comparison operation</th>
<th>operator</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>equality</td>
<td>==</td>
<td><strong>eq</strong>_</td>
</tr>
<tr>
<td>inequality</td>
<td>!</td>
<td><strong>ne</strong></td>
</tr>
<tr>
<td>less than</td>
<td>&lt;</td>
<td><strong>lt</strong></td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
<td><strong>gt</strong></td>
</tr>
<tr>
<td>less or equal</td>
<td>&lt;=</td>
<td><strong>le</strong></td>
</tr>
<tr>
<td>greater or equal</td>
<td>&gt;=</td>
<td><strong>ge</strong></td>
</tr>
</tbody>
</table>

Motivation: to compare a FlopFloat with a float.
# arithmetical operators and methods

<table>
<thead>
<tr>
<th>arithmetical operation</th>
<th>operator</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>negation</td>
<td>-</td>
<td><strong>neg</strong></td>
</tr>
<tr>
<td>addition</td>
<td>+</td>
<td><strong>add</strong></td>
</tr>
<tr>
<td>inplace addition</td>
<td>+=</td>
<td><strong>iadd</strong></td>
</tr>
<tr>
<td>reflected addition</td>
<td>+</td>
<td><strong>radd</strong></td>
</tr>
<tr>
<td>subtraction</td>
<td>-</td>
<td><strong>sub</strong></td>
</tr>
<tr>
<td>inplace subtraction</td>
<td>-=</td>
<td><strong>isub</strong></td>
</tr>
<tr>
<td>reflected subtraction</td>
<td>-=</td>
<td><strong>rsub</strong></td>
</tr>
<tr>
<td>multiplication</td>
<td>*</td>
<td><strong>mul</strong></td>
</tr>
<tr>
<td>inplace multiplication</td>
<td>*=</td>
<td><strong>imul</strong></td>
</tr>
<tr>
<td>reflected multiplication</td>
<td>*</td>
<td><strong>rmul</strong></td>
</tr>
<tr>
<td>division</td>
<td>/</td>
<td><strong>div</strong></td>
</tr>
<tr>
<td>inplace division</td>
<td>/=</td>
<td><strong>idiv</strong></td>
</tr>
<tr>
<td>reflected division</td>
<td>/=</td>
<td><strong>rdiv</strong></td>
</tr>
<tr>
<td>invert</td>
<td>~</td>
<td><strong>invert</strong></td>
</tr>
<tr>
<td>power</td>
<td>**</td>
<td><strong>pow</strong></td>
</tr>
</tbody>
</table>
The reflected (or swapped) addition happens when the first operand in + is not a FlopFloat but an ordinary number.

When \( x \) is a FlopFloat, then \( x+y \) is executed as \( x.__add__(y) \), where \( x \) is self and \( y \) is other.

For \( x + y \) when \( x \) is not a FlopFloat, but \( y \) is a FlopFloat, then \( y.__radd__(x) \) is executed.

The inplace operator allows for shorter notation, e.g.: \( += \) is defined by \( __iadd__ \).
Operator Overloading

1. OOP to count Flops
   - a flop = a floating-point operation
   - overloading arithmetical operators

2. Quaternions
   - hypercomplex numbers
   - application in computer graphics
A quaternion \( q \) (or a hypercomplex number) is

\[
q = a_0 + a_1 i + a_2 j + a_3 k,
\]

where the tuple \((a_0, a_1, a_2, a_3)\) is the coefficient vector of \( q \)
and the symbols \( i, j, \) and \( k \) satisfy

\[
i^2 = -1, \quad j^2 = -1, \quad k^2 = -1,
\]

\[
ij = k, \quad jk = i, \quad ki = j,
\]

\[
ji = -k, \quad kj = -i, \quad ik = -j,
\]

defining the multiplication of two quaternions.

The set of all quaternions is often denoted by \( \mathbb{H} \),
in honor of Sir William Rowan Hamilton who introduced them in 1843
before vector algebra was known.
Operator Overloading

1. OOP to count Flops
   - a flop = a floating-point operation
   - overloading arithmetical operators

2. Quaternions
   - hypercomplex numbers
   - application in computer graphics
Applications of Quaternions

computer graphics

Quaternions represent coordinate transformations in 3-space more compactly than matrices:

\[ q = (a_0, \mathbf{a}), \quad \mathbf{a} = (a_1, a_2, a_3). \]

Also composition of coordinate transformations goes faster with quaternions.

Quaternion multiplication \( \otimes \) with scalar, dot (\( \cdot \)), and cross product (\( \times \)):

\[ (a_0, \mathbf{a}) \otimes (b_0, \mathbf{b}) = a_0 b_0 - \mathbf{a} \cdot \mathbf{b} + a_0 \mathbf{a} + b_0 \mathbf{b} + \mathbf{a} \times \mathbf{b}. \]
Rotations in Space

Rotation about a unit vector $\mathbf{u}$ by angle $\theta$:

$$q = (s, \mathbf{v}) \quad \text{where} \quad \begin{cases} 
  s = \cos(\theta/2), \\
  \mathbf{v} = \sin(\theta/2) \mathbf{u}.
\end{cases}$$

Applying the rotation to a point $p = (x, y, z)$:

1. represent $p$ by the quaternion $P = (0, p)$,
2. compute $q \otimes P \otimes q^{-1}$.

For $q = (q_0, q_1, q_2, q_3)$, its inverse is $q^{-1} = \frac{q^*}{||q||^2}$, where

- the conjugate of $q$ is $q^* = (q_0, -q_1, -q_2, -q_3)$, and
- the magnitude of $q$ satisfies $||q||^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2$. 
Exercises

1. Overload the subtraction operator for quaternions. Also do the inplace version.

2. Provide a method `Coefficients` that returns a tuple with the coefficients of the quaternion.

3. Extend `scamul` so that you can compute the multiplication of any quaternion with any scalar multiple of $i, j, \text{ and } k$.

4. Write Python code for $\otimes$ and verify whether $q \otimes q^{-1} = 1$, for a random quaternion $q$.

5. Use operator overloading in the Python code for the class `Rational` to work with rational numbers.