We continue experimenting with simple class design.

1. Questions?

2. Thinking about Ratios of integers.

   (a) If we have gcd of two values, a and b we can compute the greatest common divisor with code similar to this:

       ```python
       def gcd(a,b):
           while a != 0:
               (a,b) = (b%a,a)
           return b if b >= 0 else -b
       ```

   (b) Recall: the __slots__ attribute of a class pre-declares the attributes of individual objects constructed by the class. You cannot add any attributes that are not mentioned in the __slots__ list. For ratios, perhaps we'd have:

       ```python
       __slots__ = ['_top', '_bottom']
       ```

   (c) Annotations. Python provides a rich collection of syntactic notes that can change how code is interpreted, called annotations. These are typically prefixed with the at-sign (@).

   (d) We learned that we can write accessor methods for our classes. If we would like to treat those accessors like attributes, we can use the @property annotation:

       ```python
       @property
       def numerator(self):
           return self._top
       ```

       Given this, we're now able to write

       ```python
       r = Ratio(10,15)
       print("Numerator is {}".format( r.numerator ))
       ```

       Note the missing parentheses! You cannot, however, assign a value to r.numerator—it's read-only.

   (e) If you do want to be able to set this pseudo attribute, you can declare a setter:

       ```python
       @numerator.setter
       def numerator(self,value):
           self._top = value
       ```
(f) Where meaningful, we can **overload** the meaning of arithmetic operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td><code>__eq__</code></td>
<td>Test for equality</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td><code>__lt__</code></td>
<td>Test for less</td>
</tr>
<tr>
<td><code>-a</code></td>
<td><code>__neg__</code></td>
<td>Negation operator</td>
</tr>
<tr>
<td><code>+a</code></td>
<td><code>__pos__</code></td>
<td>Positive operator</td>
</tr>
<tr>
<td><code>+</code></td>
<td><code>__add__</code></td>
<td>Sum of values</td>
</tr>
<tr>
<td><code>-</code></td>
<td><code>__sub__</code></td>
<td>Difference of values</td>
</tr>
<tr>
<td><code>*</code></td>
<td><code>__mul__</code></td>
<td>Product of values</td>
</tr>
<tr>
<td><code>/</code></td>
<td><code>__truediv__</code></td>
<td>Ratio of two values</td>
</tr>
<tr>
<td><code>%</code></td>
<td><code>__mod__</code></td>
<td>Remainder after division</td>
</tr>
<tr>
<td><code>//</code></td>
<td><code>__floordiv__</code></td>
<td>Whole division</td>
</tr>
</tbody>
</table>

The class annotation `@total_ordering`, imported from `functools`, will generate all comparison operations from `__lt__` and `__eq__`.

(g) Where common operators are **not** implemented, we return `NotImplemented`.

(h) The `__str__` implements the `str(r)` printable string method

(i) The `__repr__` implements the `repr(r)` representation string method

*