Computer Science 134C
Introduction to Computer Science, in Python
Lecture #17 (Classes II)
October 24

Keywords
annotation, ordering, overloading, property, setter

We continue experimenting with simple class design.

1. Questions?
2. Recall: Finishing up the `Pt` class.
3. 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:

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

   (b) Again: 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:

   ```
   __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:

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

   Given this, we're now able to write

   ```
   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*:

   ```
   @numerator.setter
   def numerator(self,value):
       self._top = value
   ```

(f) Where meaningful, we can *overload* the meaning of arithmetic operators:

`==`	`__eq__`	Test for equality
`<`	`__lt__`	Test for less
`-a`	`__neg__`	Negation operator
`+a`	`__pos__`	Positive operator
`+`	`__add__`	Sum of values
`-`	`__sub__`	Difference of values
`*`	`__mul__`	Product of values
`/`	`__truediv__`	Ratio of two values
`%`	`__mod__`	Remainder after division
`//`	`__floordiv__`	Whole division

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

⋆