1. Explain how the definition of a class supports abstraction: which part of a class definition is the interface and which is the implementation?

2. We have developed an immutable color type in Python (see the shared/examples/mar13 directory). Think about how we might improve the capabilities of the color class. You may find the documentation search box at docs.python.org/3/ useful.
   a. The \_str\_ method is called when a type must be converted to a string. How would you define this for color?
   
   b. The \_repr\_ method is called when a type must be converted to an string that returns an identical value when processed by \texttt{eval}. How would you define this for color?
c. When we compare types, python calls the \_eq\_ method of the left object on the right. For example, `c0 == c1` is really a call to `c0._eq_(c1)`. How would you write this comparison method for our `color` class?

d. One of the advantages of an immutable type is its use in dictionaries and sets. All immutable classes are aided by the definition of a \_hash\_ method that returns a consistent integer value; hash values of equal immutable objects are always the same, while unequal objects have hashes that are probably different. How would you go about constructing such a value for `color`? (Hint: numbers are immutable.)

e. The \_add\_ method takes a value, \(v\), and adds it to self. Describe how you might use this to implement equal mixing of colors. (Hint: lighten might be rewritten as `return self+color(1,1,1)`.)

*