Assignment 2: Builder Pattern

Note: I had planned to give you a choice between two different topics, but this example is SO fertile with possibilities for refactoring that I could not resist giving it to you.

Due: Friday, 4PM (EDT), 11PM (TRT)

**Motivation**

Recall robot\_bad.py. If this were a real application, it would require an endless list of attributes (perhaps hundreds), each of which would require an endless list of subcomponents.  
  
**More Motivation: the Good, the Bad, and the (Downright) Ugly!**

We saw a first cut of using the Builder Pattern to solve aspects of this problem via delegation (Good!). Gone is the long list of attributes required to create a robot, because we pushed that task over to the Builders. See robot\_with\_builder.py.

All concrete Builders inherit from an abstract superclass, so we are using SOME inheritance in our solution, but it's minimal and shallow (in other words, not terrible …)

However, the robot initializer is using lots of flag logic (BAD!) which tends to lead to (even worse) branching logic (UGLY!). One look at the **\_\_str\_\_()** method shows that our instincts were CORRECT!

**Another Wrinkle**

There is some controversy surrounding the use of the Builder Pattern in Python. Many contend that Builder is unnecessary in Python since Python allows various forms of keyword specification. A simple implementation of this, robot\_with\_keywords.com, illustrates how this might play out both in the class initializer and on the using side when instantiated.

A positive result is that the user only needs to provide relevant specifiers, not the whole list, and not in any particular order.

On the flip side, the class initializer must provide a default implementation for ALL the specifiers. Note that only the relevant components are being composed as before, but without flag logic. There are still huge problems with the **\_\_str\_\_()** method (the big and ugly decision statement), and we still have the problem of possibly unlimited multiplication of the (very disorganized) concrete classes.   
  
Another disconcerting aspect of the keyword implementation is that, as it stands now, a user could easily request attributes which are mutually exclusive or just don't make sense. Hopefully, you can see that there are some fixes for that, but it would made the code more complex

**Your mission, should you choose to accept it … (just kidding, it is, after all, an assignment)**

Armed with your knowledge of Python as well as the design principles and (limited number of) design patterns we've learned so far this semester, **design and implement an improved/refactored version of robot\_with\_builder.py**. Your implementation shall use the **Builder** pattern, along with any other useful techniques you'd like to include.