1 Administrative Topics

- None

2 Project 11

Advice:

- Be careful with the order of parameters. For example, don’t do this:
  
  ```
  s = Square()
  s.draw(0,0,0)
  ```
  
  because that third parameter is `scale` – not `zpos`. Nothing will draw!

- Drawing little objects in front of big objects is not always nice. The big object may occlude the small one, even though it seems to a human that the little object should be in front. One solution to this is to refuse to make big objects. For example, make your house by tiling lots of small rectangles rather than using one large rectangle. But, there is a drawback – this will slow down drawing and rotation.

- There is another syntax for creating strings that makes the code a bit easier to read. It uses the `%` to perform string formatting. Let me start by explaining an example
This creates a string that draws two sides of a rectangle. We can think of this as being divided into three parts: a format string, %, and the tuple containing the values. The format string tells us what the final string should look like, but uses place-holders for some of the parts. In particular, each '%f' string is a place-holder indicating that a number will appear there. Notice that there are two place-holders in the format string and two values in the final tuple (with and height). The str(width) replaces the first place-holder and str(height) replaces the second place-holder. So, if width is 40.0 and height is 45.0, then the value of \( s \) will be '(40.000000)F(90)+(45.000000)F(90)+'. You may use this notation within parentheses if you want to concatenate its result with other strings. For example, the code to create a filled rectangle string is this:

\[
\text{bw} = '\{ ' + 2*( '%f')F(90)+('%f')F(90)+'\%(width, height)') + ' ' \]

- **Aggregate Objects:** to make a super long string or to draw lots of objects? I have found that it is easier to construct a super long string. 1) For a given sub shape, you will want to begin drawing at particular (3D) orientation. Shape.draw requires us to set that initial (3D) orientation by calling orient, roll, and pitch in that order. For many orientations, it is not straightforward to determine what those three angles must be. However, if I am making a string, then I am in control of which order I apply the turns and life becomes much easier. 2) If you want to make your aggregate object orientable, then it is much easier to do that with a string (which involves relative turns) than with a set of objects. It is possible to make an aggregate object orientable, but it involves a LOT of trigonometry. So, here is some advice for making aggregate objects with strings.

- Use branching! For each component, I assume I am starting at a particular point (e.g. the lower back left corner). The first thing I do is move the turtle to the starting place of the new component. Then I use all the necessary turtle commands (well, symbols that translate to turtle commands) to draw the component. Then I end the branch and the turtle is sent back to the original position. This makes debugging a lot easier.
- To move the turtle to the starting location, I added a (parameterized) symbol “u” which is like ‘F’ but which doesn’t use a style and which picks the turtle up before going forward.

- To change the color from one component to another, I add parameterized symbols “r”, “g”, and “b”. They set the color of the appropriate channel. Note that this is done one channel at a time, so the code is a little tricky to write.

- I filled each face individually.

- I also added a parameterized symbol “l” which changes the line style. The parameter is the index into a list of my linestyles (so “(0)l” indicates the “normal” style and “(1)l” indicates the “jitter” style).