1 Administrative Topics

- We take the quiz

2 Filling Shapes

There are a three of ways we can add fill to the shapes. Assuming we have updated the Interpreter code to turn fill on when it encounters { and to turn fill off when it encounters }.

Version 1. Make a FilledSquare class derived from the Square class:

```python
class FilledSquare(Square):
    def __init__(self, distance=100, color=(1,0,0)):
        Square.__init__(self, distance, color)
        self.string = "\{" + self.string + "\}"
```

In this case, we let the parent class determine the string and we just add the fill characters. Note: this works well for basic shapes, but don’t work for composite shapes (e.g. houses, which require the components to be filled individually).

Version 2. Make a FilledSquare class derived from the Shape class:

```python
class FilledSquare(Shape):
    def __init__(self, distance=100, color=(1,0,0)):
        Shape.__init__(self, distance=distance, color=color,
```
In this case, we explicitly include the braces in the string.

**Version 3.** Add a parameter to the Square’s \texttt{\_\_init\_\_} method:

```python
class Square(Shape):
    def \_\_init\_\_(self, distance=100, color=(1,0,0), filled=False):
        bstr = 4*'F-
        if filled:
            bstr = '{' + bstr + '}'
        Shape._init_(self, distance=distance, color=color,
                     istring=bstr, angle=90)
```

This method works for both basic and composite shapes, but requires an extra parameter for filled versions of shapes.

All three methods are good.
3 Drawing Rectangles

Here is a hint: If you want to draw a rectangle with sides of arbitrary length, then use a distance of 1 and lots of F’s. To construct a string with lots of F’s, use the “multiplication” operator, i.e. if width contains an integer, you can say width * 'F’

4 Drawing a Mosaic Tile

Here is my list of recommendations for designing your tile:

• Write your code under the assumption that the tile is 1 pixel by 1 pixel. Then allow the scale to make it larger. This way, you can compute all of your positions as fractions.

• Start by drawing the outline of the tile (an unfilled square) and make sure all of your shapes end up within that square.

• You will probably want to put the same shape in the tile, but at different orientations. Take the time to draw each of your shapes and how it is drawn at certain orientations first.

I design an arrow shape and then draw it with the same location and orientation as my triangle. This made it easy for me to visualize how to place my triangle at specific locations and orientations.

Here is the code:

```
t = shapes.FilledTriangle( distance = 100, color = 'red’ )
a = shapes.Arrow( distance = 30 )
t.draw( 0, 0, orientation = 0 )
a.draw( 0, 0, orientation = 0 )
t.setColor( 'blue’ )
t.draw( 100, 100, orientation = 90 )
a.draw( 100, 100, orientation = 90 )
t.setColor( 'yellow’ )
t.draw( 0, 100, orientation = 180 )
a.draw( 0, 100, orientation = 180 )
t.setColor( 'green’ )
```
The output is in Fig. 1.

```
t.draw( −100, 0, orientation = 270 )
a.draw( −100, 0, orientation = 270 )
```

Figure 1: Triangles at various orientations.

Now I can draw a tile with the 4 triangles forming an inner square. My design is to have the edges of the triangle be 1/3 of the edge length of the square.

But first, I draw a square as the outline of the tile (to help me position things). I will remove this once I have drawn the filled shapes in it. This results in an incredibly boring picture, but it will make my life easier if I can see the boundary of the tile. I think this step is so important that I am including code for it an a picture of it (see Fig. 2).

```
def tile(x,y,scale):
    s = shapes.Square( distance = 1 )
    s.draw( x, y, scale, orientation=90 )
```
Then I start by creating my Triangle. Note that I am going to use a distance of 1, and will use the scale to get the distance just right. I then draw my first triangle, which will be the bottom triangle (which points down). By looking at Fig 1, I see that if I want to draw a triangle like that, then I need to specify the position of the upper left corner and the orientation of 0. So I do that (see Fig 3).

```python
def tile(x, y, scale):
    s = shapes.Square(distance=1)
    s.draw(x, y, scale, orientation=90)
    t = shapes.FilledTriangle(distance=1, color='red')
    t.draw(x+0.33*scale, y+0.33*scale, 0.33*scale, orientation=0)
```

Now that I know I have the sizes right and my approach will work, I can work out the positions of the remaining three triangles. And I will be so bold as to draw all three of them before testing my code. Hurrah! It works (see Fig 4).

```python
def tile(x, y, scale):
    s = shapes.Square(distance=1)
    s.draw(x, y, scale, orientation=90)
    t = shapes.FilledTriangle(distance=1, color='red')
```
Figure 3: The outline of the tile with the bottom triangle.

\begin{verbatim}
t.draw( x+0.33*scale, y+0.33*scale, 0.33*scale, orientation=0 )
t.draw( x+0.67*scale, y+0.33*scale, 0.33*scale, orientation=90 )
t.draw( x+0.67*scale, y+0.67*scale, 0.33*scale, orientation=180 )
t.draw( x+0.33*scale, y+0.67*scale, 0.33*scale, orientation=270 )
\end{verbatim}
Figure 4: The outline of the tile with all four triangles.