CS151 Fall 2012 Lecture 28

Stephanie R Taylor

November 9, 2012

1 Administrative Topics

- We take the quiz

2 Symbol tables and inheritance

When a child class is derived from a parent class, its symbol tables entries are "copied" from the parent class's symbol table (i.e. the values are pointers to the corresponding entries in the parent's table). Any method that overrides the parent's method has a value that points to the new function (instead of an arrow to the parent table). Consider the Shape class from the project. It has methods __init__ and draw (in addition to a bunch of mutators that I will ignore for this example). The Square class is derived from the Shape class, and it overrides the __init__ method. So we have these two symbol tables as shown in Fig. 1

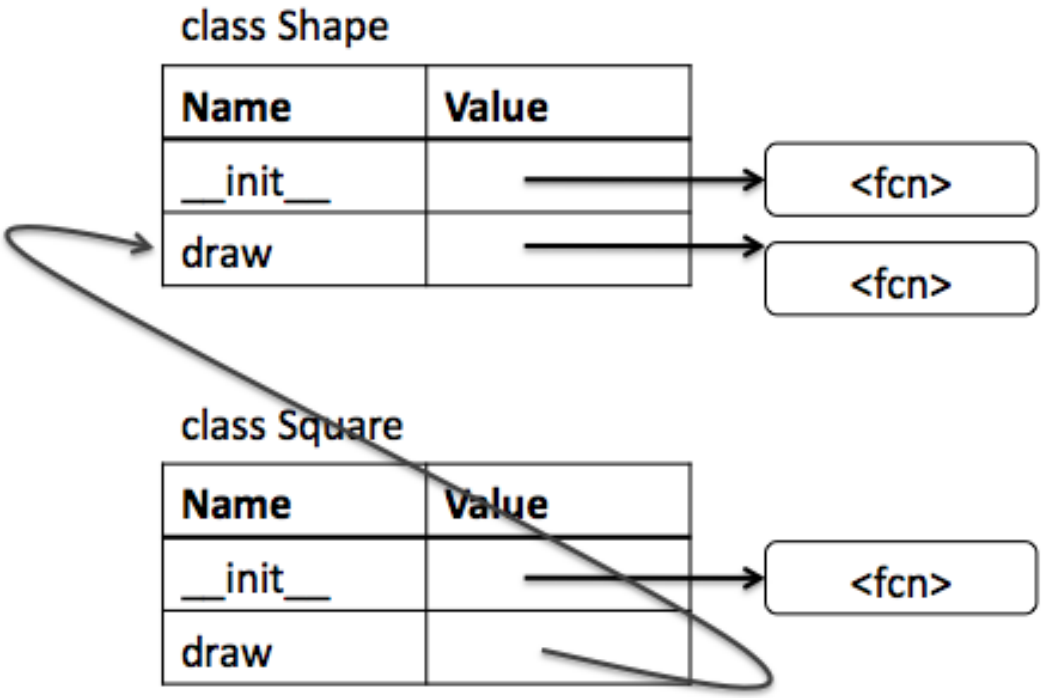

Figure 1: Symbol tables for parent class Shape and child class Square

When we make an instance of the Square class (i.e. a Square object), the object's symbol table contains entries pointing to the Square table's entries. It also contains data field (e.g. distance). If we were to call the draw method on the object, Python would look in the object's symbol table for the draw entry, and follow the arrow up to the Square table's draw entry, and up to the Shape table's draw entry until it finally finds the function definition. See Fig. 2 for the arrows.

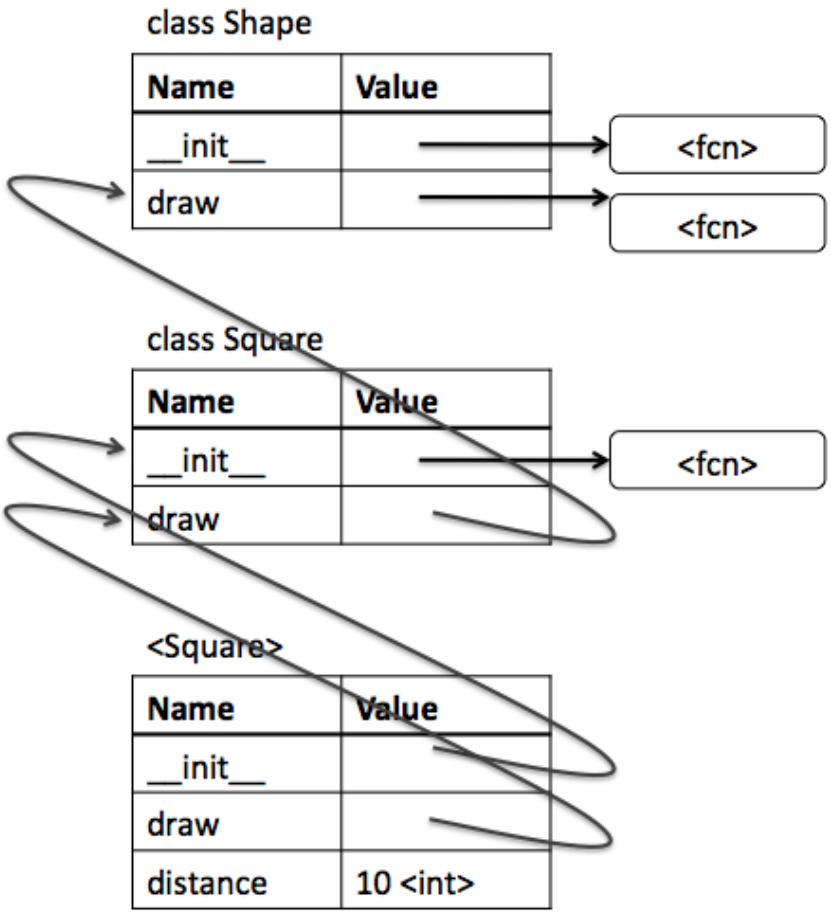

Figure 2: Symbol tables for parent class Shape, child class Square, and a Square object

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' )
```

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

The output is in Fig. 3.

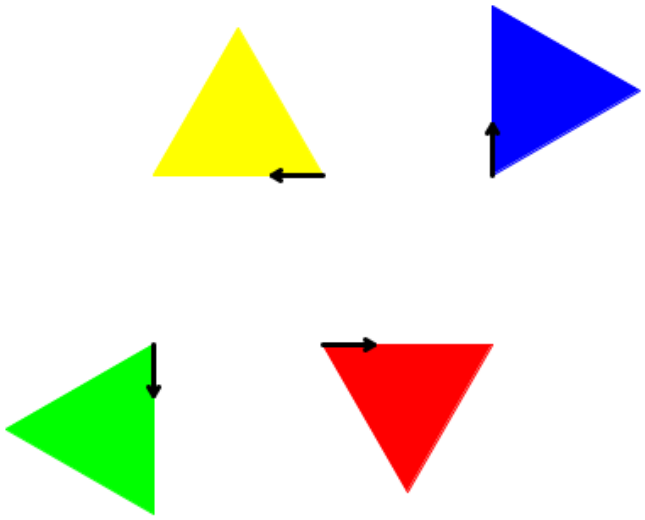

Figure 3: Triangles at various orientations.

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. 4).

```
def tile(x,y,scale):
    s = shapes.Square( distance = 1 )
    s.draw( x, y, scale, orientation=90  )
```

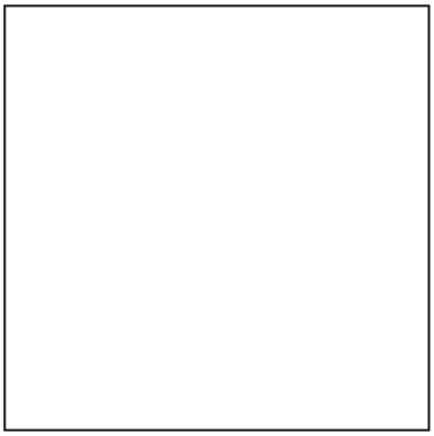

Figure 4: The outline of my tile.

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 3, 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 5).

```
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 6).

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

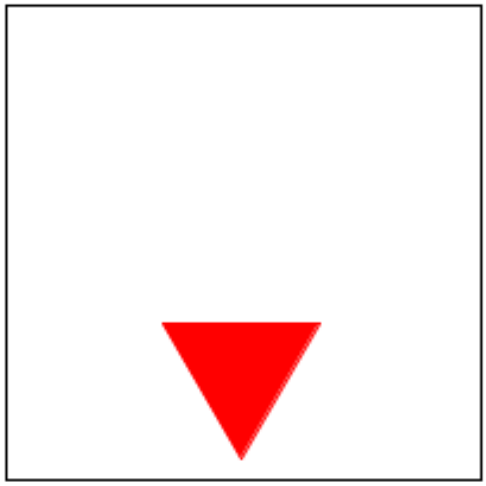

Figure 5: The outline of the tile with the bottom triangle.

```
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 )
```

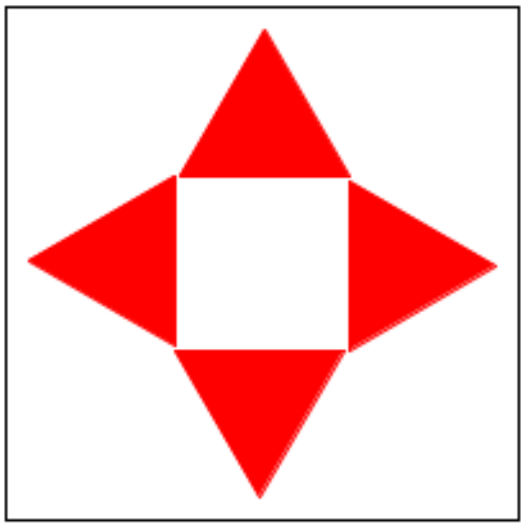

Figure 6: The outline of the tile with all four triangles.