1 Administrative Topics

- Next Thursday is the Celebration of CS at Colby. Afternoon sessions. Please come!

2 Making Classes to Draw objects

We continue the example we began on Monday. In Monday’s notes, I list two designs. In class on Monday we went over only the “alternate design”. Today, we will go over the favored design and extend it to make the trees reflect the seasons.

Drawing a Tree

Reminder of our goal from Monday: Let’s design a class to draw a very simplified tree (a circle on top of a rectangle). We want it to be initialized with similar information as the simple objects (namely, position and size information). In our case, we will make it simpler - the caller can supply numbers instead of Point objects. We also need a draw method that will take a GraphWin object as a parameter and draw the objects.

What do we need for local variable and what needs to be a field (i.e. what needs to belong in the object’s symbol table)? In general, something should
be stored in a field if we need to access that data in more than one method call (i.e. you need the data to stay there when you call the same method twice or you need two different methods to access it).

For this class, I propose we use two fields - one for the circle (the leafy part) and one for the rectangle (the trunk).

```python
class Tree:
    def __init__(self, topCenterX, topCenterY, radius):
        self.leafPart = gr.Circle(gr.Point(topCenterX, topCenterY), radius)
        self.leafPart.setFill('green')
        self.trunk = gr.Rectangle(gr.Point(topCenterX−radius/8.0, topCenterY + radius),
                                   gr.Point(topCenterX+radius/8.0, topCenterY + radius*3))
        self.trunk.setFill('brown')

    def draw(self, win):
        self.trunk.draw(win)
        self.leafPart.draw(win)

win = gr.GraphWin("Demo", 500, 500)
t = Tree(250, 200, 50)
t.draw(250, 200, 50)
win.getMouse()
win.close()
```

Let’s also think about the symbol tables. After the code has finished (but before everything is destroyed, we have 3 symbol tables that we are interested in (I am for now ignoring the symbol tables for the graphics module and the graphics objects).

The main symbol table is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphics</td>
<td>&lt;module&gt;</td>
</tr>
<tr>
<td>win</td>
<td>&lt;GraphWin&gt;</td>
</tr>
<tr>
<td>Tree</td>
<td>&lt;Class Tree&gt;</td>
</tr>
<tr>
<td>t</td>
<td>&lt;Tree&gt;</td>
</tr>
</tbody>
</table>

The Tree class’s symbol table has just the methods:
The Tree object’s symbol table has the methods and the data:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
<td>&lt;function&gt;</td>
</tr>
<tr>
<td>draw</td>
<td>&lt;function&gt;</td>
</tr>
<tr>
<td>leafPart</td>
<td>&lt;Circle&gt;</td>
</tr>
<tr>
<td>trunk</td>
<td>&lt;Rectangle&gt;</td>
</tr>
</tbody>
</table>

Making our Tree Seasonal

Now let’s add method that allows us to change our Tree’s color, depending on the season. We can do this easily because we have fields with the Graphics objects in them. We can keep our current methods without changing them and simply add a changeSeason method that will change the color of the leafy part, depending on the season.

```python
class Tree:
    def __init__(self, topCenterX, topCenterY, radius):
        self.leafPart = gr.Circle(gr.Point(topCenterX, topCenterY), radius)
        self.leafPart.setFill('green')
        self.trunk = gr.Rectangle(gr.Point(topCenterX-radius/8.0, topCenterY + radius),
                                  gr.Point(topCenterX+radius/8.0, topCenterY + radius*3))
        self.trunk.setFill('brown')

    # Change the color of the tree, depending on the season
    # season should be 'fall', 'summer', 'spring', or 'winter'
    def changeSeason(self, season):
        if season == 'fall':
            self.leafPart.setFill('orange')
        elif season == 'summer':
            self.leafPart.setFill('green')
        elif season == 'spring':
            self.leafPart.setFill('yellow')
```

3
elif season == 'winter':
    self.leafPart.setFill('white')

def draw(self, win):
    self.trunk.draw(win)
    self.leafPart.draw(win)

win = gr.GraphWin("Demo", 500, 500)
t = Tree(250, 200, 50)
t.changeSeason('spring')
t.draw(win)
time.sleep(0.25)
t.changeSeason('summer')
time.sleep(0.25)
t.changeSeason('fall')
time.sleep(0.25)
t.changeSeason('winter')
win.getMouse()
win.close()

Also notice that we use the Python package `time` to make a pause in our simulation.

time.sleep(0.25)

pauses the program for 0.25 seconds. This is really useful for animation!

Notice that our init method didn’t change at all. It started with a green tree. But I suggest we leverage the existence of the change_seasons method and make it start out by default as a “summer” tree. We can do that like this:

def __init__(self, topCenterX, topCenterY, radius, 
             season = 'summer '):
    self.leafPart = gr.Circle(
        gr.Point(topCenterX, topCenterY), 
        radius)
    self.changeSeason(season)
    self.trunk = gr.Rectangle(
        gr.Point(topCenterX-radius/8.0, 
                  topCenterY + radius), 
        gr.Point(topCenterX+radius/8.0, 
                  topCenterY + radius*3))
    self.trunk.setFill('brown')
Our Trees are Really Ents

Since your next project involves animation of moving objects, we should practice making things move. Ents are trees that move. So, let’s make our Tree class an Ent class and add a move method. To move our tree, we need to move each part of the tree. Each Graphics object has a method called move that takes as input the distance in x to move and the distance in y to move. We need to use that method and since we are trying to make our complex objects have similar methods to the simple graphics objects, we should copy them and make the same method, i.e.

```python
# move the tree dx pixels in the x-direction and
# dy pixels in the y-direction
def move( self, dx, dy ):
    self.leafPart.move( dx, dy )
    self.trunk.move( dx, dy )
```

We can make our tree move with test code like this:

```python
# make it walk
for i in range(4):
    time.sleep( 0.25 )
    t.move( 20, 0 )
```