1 Administrative Topics

- There is a quiz on Friday.
- The project is due on Friday this week instead of Tuesday.

2 Memory model of graphics code

The goal for today’s class is to gain a mental picture of the objects and lists that will be used in Project 6. In the project, we are calling the list of shapes “shape groups”. The code in these notes calls the “components”. They are the same thing.

2.1 Code

The component (shape group) code is

```python
# Stephanie Taylor
# shape_group.py
# A "shape group" is a list of graphics objects.
import graphics as gr

# draw a list of graphics object in the given window
```
def draw(objlist, win):
    for item in objlist:
        item.draw(win)

# draw a funky ball at (x,y).
def ball_init(x, y, scale):
    shapes = []
    b = gr.Circle(gr.Point(x, y), 20*scale)
    b.setFill("blue")
    b.setOutline('yellow')
    b.setWidth(4*scale)
    shapes.append(b)
    b = gr.Circle(gr.Point(x + 15*scale, y), 20*scale)
    b.setFill("blue")
    b.setOutline('yellow')
    b.setWidth(4*scale)
    shapes.append(b)
    print "printing shapes"
    print shapes
    return shapes

def beach_init(x, y, scale):
    shapes = []
    r = gr.Rectangle(gr.Point(x, y),
                     gr.Point(x + 300*scale, y + 130*scale))
    r.setFill(gr.color_rgb(210, 180, 140))
    shapes.append(r)
    return shapes
The main scene code is

```python
# Stephanie Taylor
# scene1.py

import graphics as gr
import component as cmp

def main():
    win = gr.GraphWin( "Beach Scene", 300, 300, False )

    ball = cmp.ball_init( 150, 150, 1 )
    beach = cmp.beach_init( 0, 170, 1 )
    scene_list = [beach, ball]

    for part in scene_list:
        cmp.draw( part, win )

    win.update()

    # close
    win.getMouse()
    win.close()

if __name__ == "__main__":
    main()
```

3
2.2 Stepping through the code

We begin by executing the top-level code. That puts `gr`, `cmp`, and `main` in the symbol table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gr</code></td>
<td></td>
</tr>
<tr>
<td><code>cmp</code></td>
<td></td>
</tr>
<tr>
<td><code>main</code></td>
<td></td>
</tr>
</tbody>
</table>
Then line 22 of the scene module calls the main function. A new symbol table appears.

The first line of code in `main` is

```python
win = gr.GraphWin( "Beach Scene", 300, 300 )
```

After executing this line, the `main` function’s symbol is updated to include an arrow to the graphics window object:
The next line of code in main is

```
ball = cmp.ball_init( 150, 150, 1 )
```

We will step through the call to `cmp.ball_init`. A new symbol table is created for it. The parameter values are entered into the table. The first line of code initializes the variable `shapes` to an empty list, so that is also added to the table:
The next few lines of `ball_init` create the first circle and change its properties.

```python
b = gr.Circle( gr.Point(x,y), 20*scale )
b.setFill( "blue" )
b.setOutline( 'yellow' )
b.setWidth( 4*scale )
```

The first circle is created and stored with the name `b`:
The next line of `ball_init` is

```
shapes.append(b)
```

which updates the contents of the `shapes` list:

```
main (top-level for scene1.py)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr</td>
<td></td>
</tr>
<tr>
<td>cmp</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td></td>
</tr>
</tbody>
</table>

main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>win</td>
<td></td>
</tr>
</tbody>
</table>

ball_init

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>150 &lt;int&gt;</td>
</tr>
<tr>
<td>y</td>
<td>150 &lt;int&gt;</td>
</tr>
<tr>
<td>scale</td>
<td>1 &lt;int&gt;</td>
</tr>
<tr>
<td>shapes</td>
<td>1 &lt;int&gt;</td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

[Diagram showing `shapes.append(b)` and the updated `shapes` list]
The next few lines of `ball_init` create the second circle and change its properties.

```python
b = gr.Circle( gr.Point(x+15*scale,y), 20*scale )
b.setFill( "blue" )
b.setOutline( 'yellow' )
b.setWidth( 4*scale )
```

The second circle is created and stored with the name `b`:
The next line of `ball_init` is

```
shapes.append(b)
```

which updates the contents of the `shapes` list:

```
[<Circle>, <Circle>]
```
The value of the `shapes` list is returned to the caller (`main`). That code adds it to the symbol table, using the name `ball`:
And we redraw the memory in a more compact way:
The next line of code in main is

```python
beach = cmp.beach_init(0, 170, 1)
```

cmp.beach_init is called. It returns a list of objects and the arrow to that list is stored with the name beach:
And we redraw the memory in a more compact way:
The next line of code in `main` is

```python
scene_list = [beach, ball]
```

which makes a list of lists. Below, we draw the arrows associated with `scene_list` in blue, to make it easier for the eye to follow the different arrows.
Next in `main` is the loop

```python
for part in scene_list:
cmp.draw(part, win)
```

The first time through the loop, `part` refers to the ball list (we use a red arrow for this one):

The code in the loop executes the `draw` function in `component.py`. This function will in turn loop through each object in the list, calling that object’s `draw` method.
The second time through the loop, `part` refers to the beach list (we use a red arrow for this one):

The code in the loop executes the `draw` function in `component.py`. This function will in turn loop through each object in the list, calling that object’s `draw` method.

Then we move on to the code

```python
    win.getMouse()
    win.close()
```

which waits for the mouse to be clicked, and then closes the window. Then the main function is done, and all of the code in the module has been executed.

Bye!
3 General debugging advice

- Expected number of arguments: When you are calling a method on an object, the object itself is hidden argument. So that means if you have a method such as `Circle.getCenter()`, there are 0 parameters between the parentheses. But the Circle object itself is a parameter. So if you write code like this:

  ```python
c = graphics.Circle(graphics.Point(100,100), 5)
c.getCenter(1)
```

Python will tell you that it expected 1 argument and got 2.

- What is wrong with

  ```python
  if random.random < 0.2:
  ```

  It is missing the parentheses, so the random function doesn’t execute. It should be

  ```python
  if random.random() < 0.2:
  ```

  In the first case, we are comparing a pointer to the random function to the number 0.2. In the second case, we are comparing the output of the random function to the number 0.2. We want the second case.