CSC 110: Week 9, Lecture 1

Dr. Bodine

Last week…

- Objects!
- Object-oriented programming
- Class definitions
- Abstraction and encapsulation
- Inheritance

This week…

- Continue learning about OOP
- Graphics as an example of objects and OOP
- Basic graphics in python
  - Using a specific library….may learn about others if we have time...

Objects and OOP

- Objects:
  - Instances of a class
  - Active data types containing information (instance variables) and how to handle the information (methods)

- Object-oriented programming:
  - Programmer defines objects (data and how it will be handled)
  - Encapsulate details in definition of objects (often classes)

Python classes

```
class <class name> (<what it inherits from>):
    def __init__(self,<any other parameters>):
        <statements>
        <usually assign instance variables >

    <other statements, like defining methods>
```

Examples of classes we wrote:

- Student class
- Shape2d class
  - Circle class
  - Rectangle class

How does this relate to our topic this week?

- Graphics are a good way to learn about objects and OOP
  - Natural connection to real world objects
  - Understanding inheritance

- In particular, we will get a handle on methods and the concept of a complicated class structure
  - Hides a lot of the details, let's us easily do things…

Before getting started…

- Need to start with a graphics library

- For simplicity we will use the one in the book:
  - http://mcsp.wartburg.edu/zelle/python/ppics2/code/graphics.py
  - Adapted from the standard python Tkinter
  - Can download from this week's module on Canvas

- Need to save this in your code folder
  - We will import it to gain access to the files

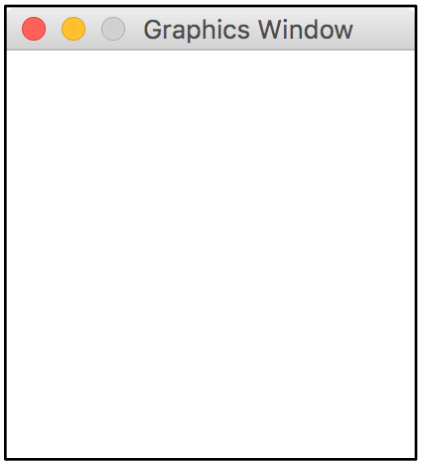
Getting started…

- Need to import the graphics library
  - Start with `import graphics`
- Recall in order to access anything within graphics.py we need to use the dot notation
  - `graphics.Point(<x-value>,<y-value>)` will make a point
- What if we know we want to use a lot of things from graphics.py?
  - Can import with `from graphics import *`
  - Use this wisely as you can easily run into namespace collisions

Graphics window

- Before we can draw anything we need to have a space to draw
  - This is called a Graphics Window
- `GraphWin(title, width, height)`

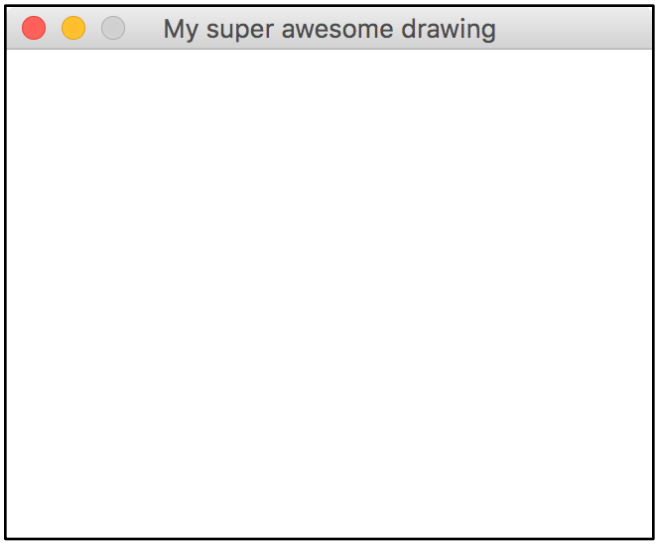
```
>>> import graphics
>>> mywin = graphics.GraphWin()
```



Graphics window 2.0

- Using the default constructor gave us a 200x200 pixel window
- But what if we want a bigger window or to give it a cool name?

```
>>> mywin = graphics.GraphWin("My super awesome drawing",320,240)
```



Now, how do I draw in it?

- Well, we can specify pixel by pixel what color to draw…
  - but that sounds like a lot of work.

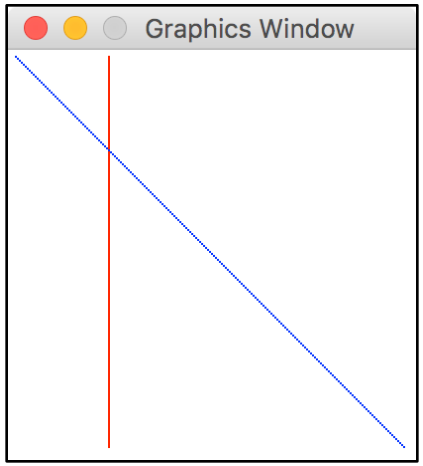
- We can also use built-in shapes that are specified via classes!
  - Have attributes to do bookkeeping for us.

- Upshot: We create objects, then we tell those objects to draw themselves!

Plotting points…

- Graphics window plotting method: `plot(x,y,color)`

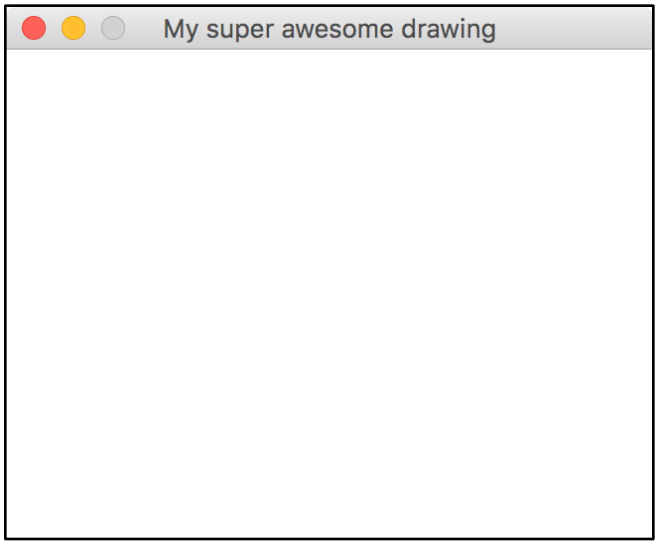
```
>>> mywin = graphics.GraphWin()
>>> for i in range(0,200):
        mywin.plot(50,i,"red")


>>> for i in range(0,200):
        mywin.plot(i,i,"blue")
```



Start with a point

`Point(x,y)`

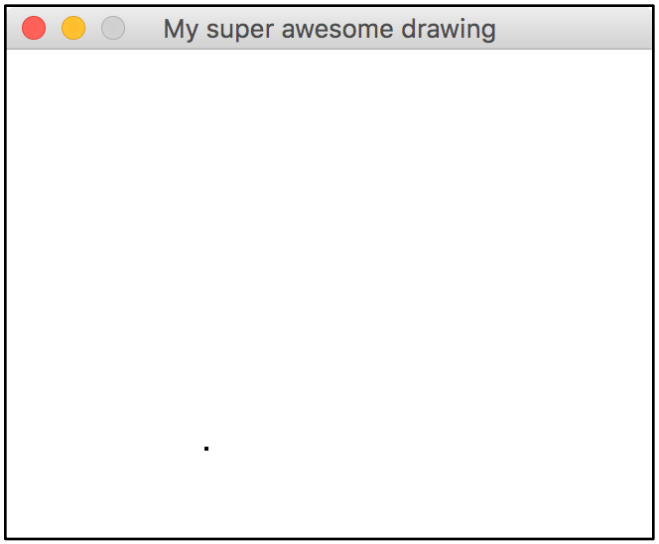
```
>>> mypoint = graphics.Point(100,200)
>>> mypoint.getX()
100
>>> mypoint.getY()
200
```



Start with a point

`Point(x,y)`

```
>>> mypoint = graphics.Point(100,200)
>>> mypoint.getX()
100
>>> mypoint.getY()
200
>>> mypoint.draw(mywin)
```



How exactly does that work?

```
mypoint.draw(mywin)
```

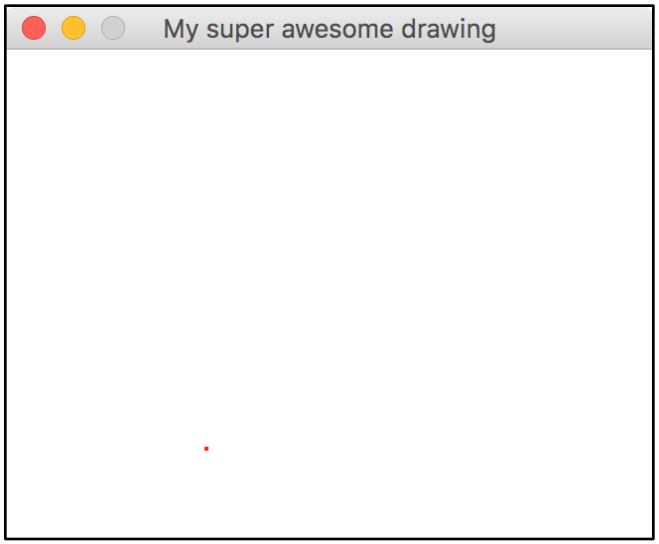
- mypoint is an instance of the Point class
- draw is a method that takes a parameter
- mywin is used as a parameter -> telling mypoint where to draw itself

- How does all that work behind the scenes???
  - We don’t need to know -> Encapsulation!!!

Start with a point

`Point(x,y)`

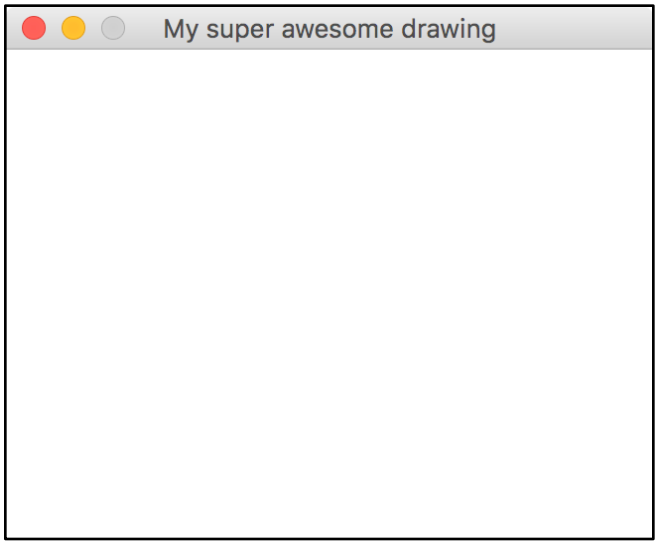
```
>>> mypoint = graphics.Point(100,200)
>>> mypoint.getX()
100
>>> mypoint.getY()
200
>>> mypoint.draw(mywin)
>>> mypoint.setFill('red')
```



Start with a point

`Point(x,y)`

```
>>> mypoint = graphics.Point(100,200)
>>> mypoint.getX()
100
>>> mypoint.getY()
200
>>> mypoint.draw(mywin)
>>> mypoint.setFill('red')
>>> mypoint.undraw()
```



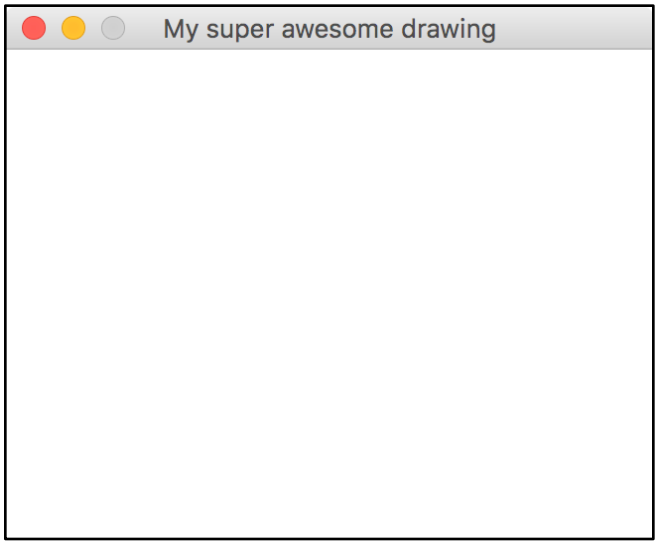
Points as objects

- mypoint was an object: an instance of the Point class
- Did mypoint automatically draw?
- How did mypoint know where to draw?
- Did drawings automatically update?

Start with a circle

`Circle(centerPoint, radius)`

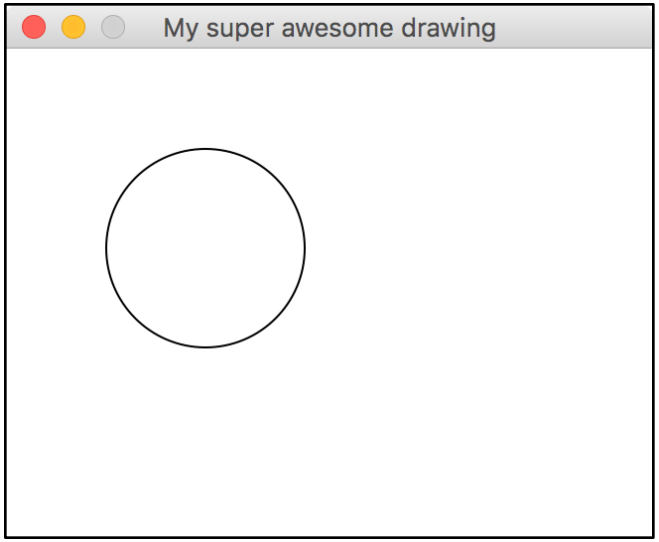
```
>>> mywin = graphics.GraphWin("My super awesome drawing",320,240)
>>> mycircle = graphics.Circle(graphics.Point(100,100),50)
>>>
>>> mycircle.getCenter()
<graphics.Point object at 0x105971e10>
>>> centerpt = mycircle.getCenter()
>>> centerpt
<graphics.Point object at 0x1059729b0>
>>>
>>> centerpt.getX()
100.0
>>> centerpt.getY()
100.0
>>> mycircle.getCenter().getX()
100.0
```



Start with a circle

`Circle(centerPoint, radius)`

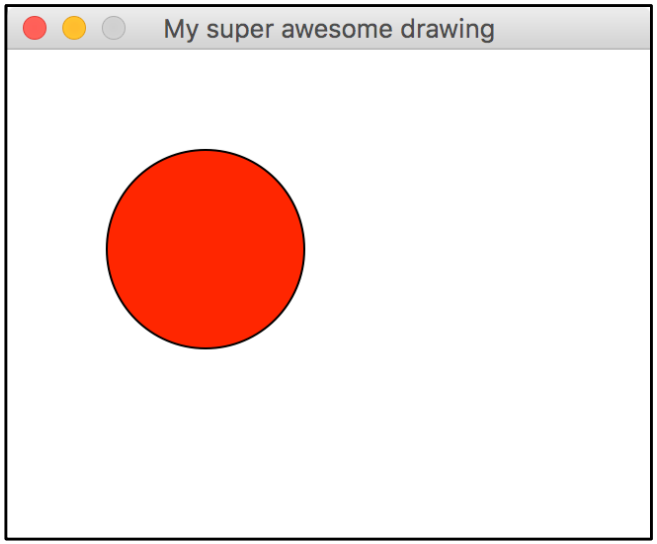
```
>>> mywin = graphics.GraphWin("My super awesome drawing",320,240)
>>> mycircle = graphics.Circle(graphics.Point(100,100),50)
>>>
>>> mycircle.getCenter()
<graphics.Point object at 0x105971e10>
>>> centerpt = mycircle.getCenter()
>>> centerpt
<graphics.Point object at 0x1059729b0>
>>>
>>> centerpt.getX()
100.0
>>> centerpt.getY()
100.0
>>> mycircle.getCenter().getX()
100.0
>>> mycircle.draw(mywin)
```



Start with a circle

`Circle(centerPoint, radius)`

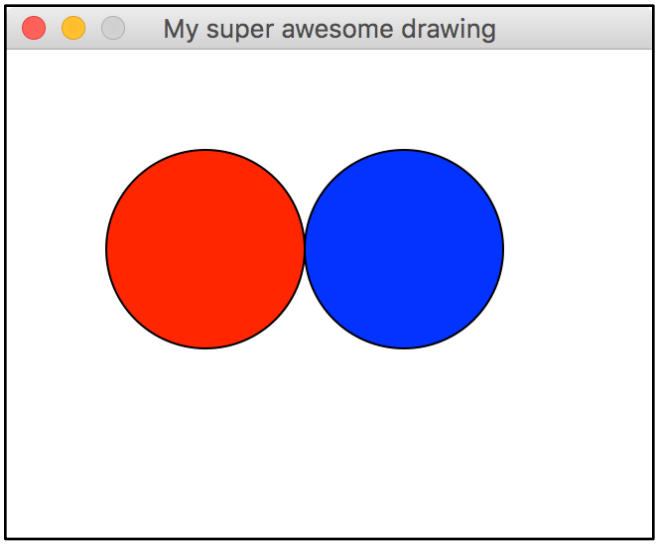
```
>>> mywin = graphics.GraphWin("My super awesome drawing",320,240)
>>> mycircle = graphics.Circle(graphics.Point(100,100),50)
>>>
>>> mycircle.getCenter()
<graphics.Point object at 0x105971e10>
>>> centerpt = mycircle.getCenter()
>>> centerpt
<graphics.Point object at 0x1059729b0>
>>>
>>> centerpt.getX()
100.0
>>> centerpt.getY()
100.0
>>> mycircle.getCenter().getX()
100.0
>>> mycircle.draw(mywin)
>>> mycircle.setFill("red")
```



Start with a circle

`Circle(centerPoint, radius)`

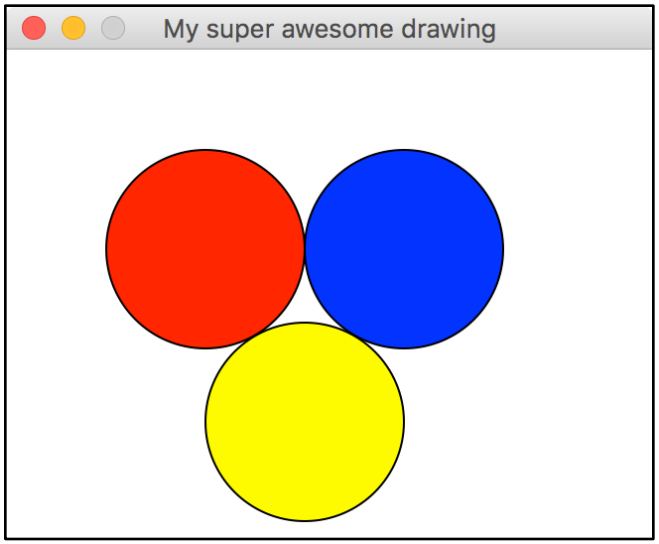
```
>>> mycircle2 = mycircle.clone()
>>> mycircle2.move(100,0)
>>> mycircle2.setFill("blue")
>>> mycircle2.draw(mywin)
>>>
```



Start with a circle

`Circle(centerPoint, radius)`

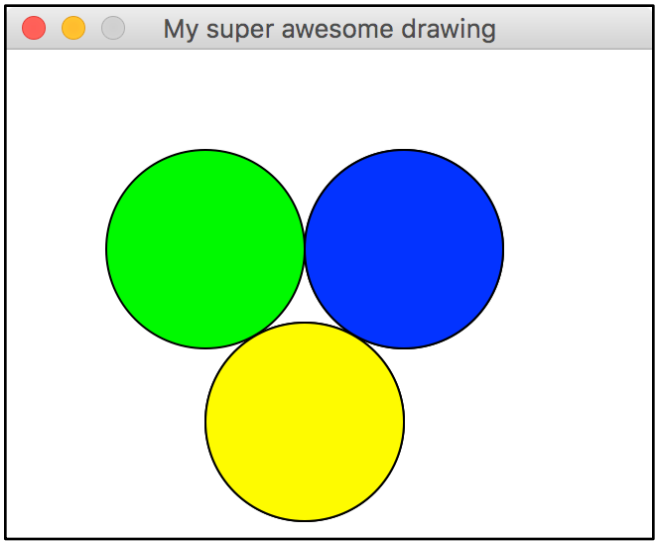
```
>>> mycircle2 = mycircle.clone()
>>> mycircle2.move(100,0)
>>> mycircle2.setFill("blue")
>>> mycircle2.draw(mywin)
>>>
>>> mycircle3 = mycircle.clone()
>>> mycircle3.move(50,87)
>>> mycircle3.setFill("yellow")
>>> mycircle3.draw(mywin)
>>>
```



Start with a circle

`Circle(centerPoint, radius)`

```
>>> mycircle2 = mycircle.clone()
>>> mycircle2.move(100,0)
>>> mycircle2.setFill("blue")
>>> mycircle2.draw(mywin)
>>>
>>> mycircle3 = mycircle.clone()
>>> mycircle3.move(50,87)
>>> mycircle3.setFill("yellow")
>>> mycircle3.draw(mywin)
>>>
>>>
>>> mycircle.setFill("green")
```



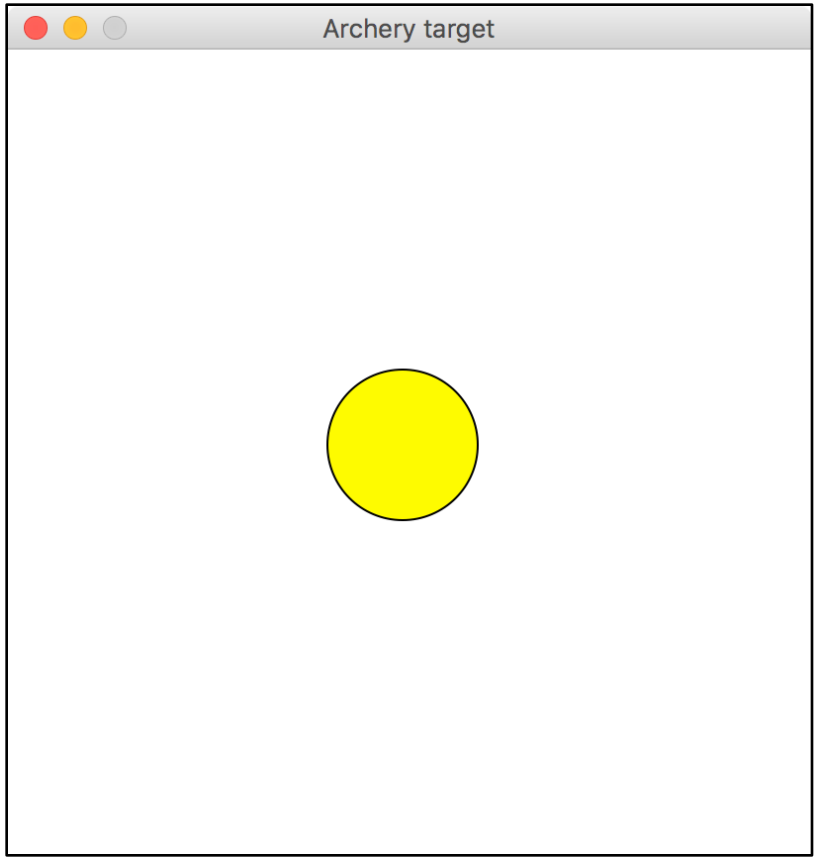
What did we learn?

- We can CLONE objects to avoid having to duplicate code
- We can then alter clones to look like we want them
  - Change color
  - Change location
- When we update the original, the clones do not update…
  - They were cloned from a specific state

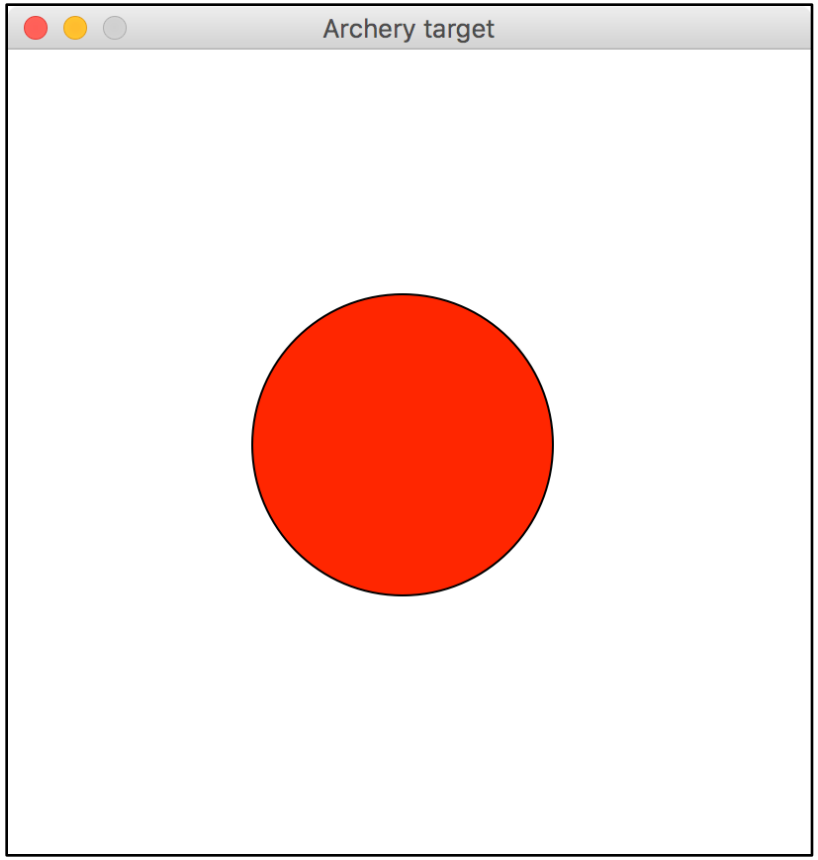
Let’s have some fun with it!

- Let’s try to draw an archery target with 5 rings: yellow, red, blue, black and white in order from inside to outside.
- Each ring will have the same width as the radius of the innermost (yellow) circle.
- Display the target in a square window of 400x400 pixels

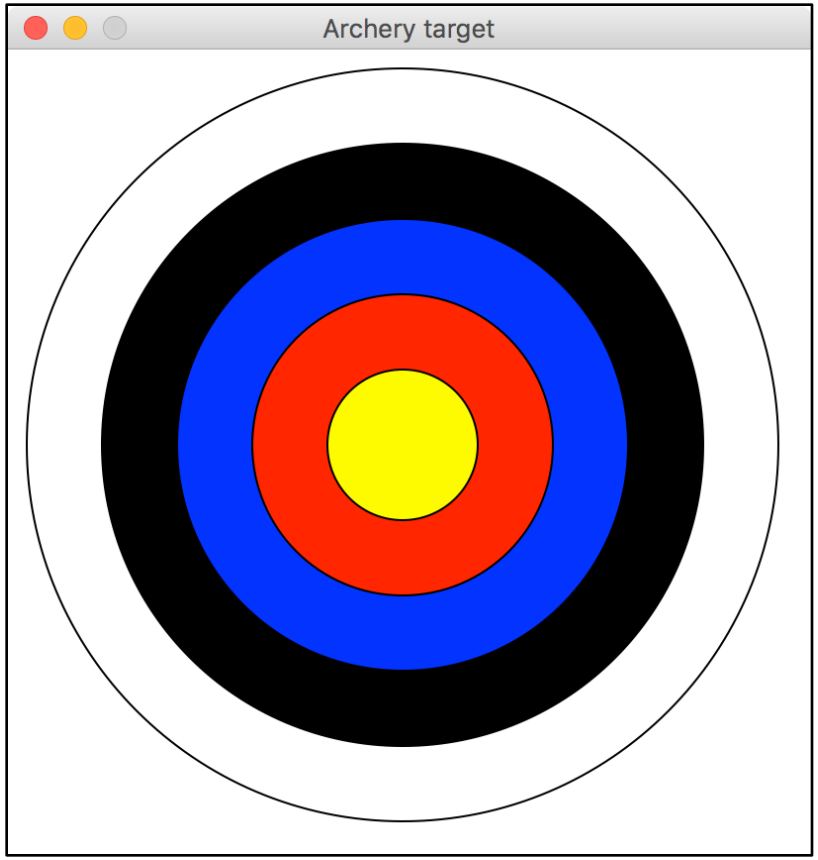
```
archwin = graphics.GraphWin("Archery target",400,400)

center = graphics.Point(199,199)
radius = 380/10

yellowcircle = graphics.Circle(center,radius)
yellowcircle.setFill("yellow")
yellowcircle.draw(archwin)
```



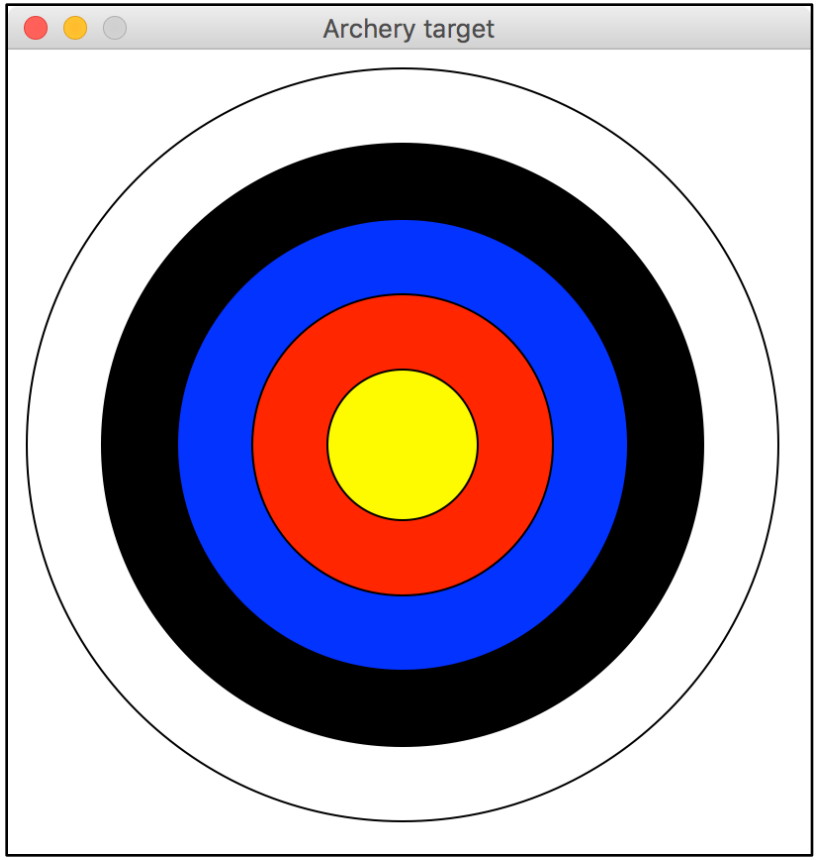
```
archwin = graphics.GraphWin("Archery target",400,400)

center = graphics.Point(199,199)
radius = 380/10

yellowcircle = graphics.Circle(center,radius)
yellowcircle.setFill("yellow")
yellowcircle.draw(archwin)

redcircle = graphics.Circle(center,2*radius)
redcircle.setFill("red")
redcircle.draw(archwin)
```



```
archwin = graphics.GraphWin("Archery target",400,400)

center = graphics.Point(199,199)
radius = 380/10

whitecircle = graphics.Circle(center,5*radius)
whitecircle.draw(archwin)

blackcircle = graphics.Circle(center,4*radius)
blackcircle.setFill("black")
blackcircle.draw(archwin)

bluecircle = graphics.Circle(center,3*radius)
bluecircle.setFill("blue")
bluecircle.draw(archwin)

redcircle = graphics.Circle(center,2*radius)
redcircle.setFill("red")
redcircle.draw(archwin)

yellowcircle = graphics.Circle(center,radius)
yellowcircle.setFill("yellow")
yellowcircle.draw(archwin)
```



```
archwin = graphics.GraphWin("Archery target",400,400)

center = graphics.Point(199,199)
radius = 380/10

#Create the circles and set fill colors
yellowcircle = graphics.Circle(center,radius)
yellowcircle.setFill("yellow")

redcircle = graphics.Circle(center,2*radius)
redcircle.setFill("red")

bluecircle = graphics.Circle(center,3*radius)
bluecircle.setFill("blue")

blackcircle = graphics.Circle(center,4*radius)
blackcircle.setFill("black")

whitecircle = graphics.Circle(center,5*radius)

#Draw the circles in the correct order
whitecircle.draw(archwin)
blackcircle.draw(archwin)
bluecircle.draw(archwin)
redcircle.draw(archwin)
yellowcircle.draw(archwin)
```



Upshot about ordering…

- The object that will be in the front view (obscuring any other objects that share coordinates) is the last object we draw.

- We can define object in any order we would like because it is the order we DRAW them in that matters

A thought on coordinates…

- We can do everything pixel by pixel…but then I have to KNOW ahead of time how big of a window will be used
- What if I don’t know?
- It would be nice if there was a way to define coordinates that didn’t rely on pixel by pixel definition
- Multiple approaches to handling this…