VPython Part 2

Many of these slides come from here

- http://www.glowscript.org/docs/VPythonDocs/index.html
- https://www.glowscript.org/docs/VPythonDocs/VPython_Intro.pdf
- https://www.glowscript.org/docs/VPythonDocs/videos.html

Animation!

- Often we want animation!
- Objects that move in time.
- When creating a shape you can have it rotate by right-clicking. But what if you want it to move?
- I want the shape to move, but how do I refer to it?
- I need to give it a name?

- movingBox=box(pos=vector(-5, 0,0))
- To get the box to move from left to right what needs to change?

Animate!

```
movingBox=box(pos=vector(-5, 0,0))
# Move a box across the scene in a straight line:
while movingBox.pos.x < 5:
        movingBox.pos.x = movingBox.pos.x + 0.05
```

Remember colon

What happens when we run this?

Why don’t we see it moving?

It’s too fast to see what’s happening! What can we do?

Wait

- We want to slow it down so we could see what's happening each time in the loop.

  rate(frequency)
- Stops computations until 1/frequency seconds after the previous call to rate().
- Examples:
  - rate(50) = 1/50 = wait until 0.02 second has elapsed (or do the loop 50 times per second).
  - rate(1) = loop will execute at a maximum of 1 time per second (1 frame per second)

(Another option you can use is sleep(1) to sleep for 1 second)

Let’s try again!

```
movingBox=box(pos=vector(-5, 0,0))
# Move a box across the scene in a straight line:
while movingBox.pos.x < 5:
	rate(50)
	movingBox.pos.x += 0.05
```

What happened?

What happens if we switch rate to 1 second instead?

Another Example: Animation

```
my_sphere=sphere(pos=vector(0,0,0), radius=0.25, color=color.green)
i=1
while(i<=5):
        rate(1) #show 1 frame per second
        my_sphere.pos.x= my_sphere.pos.x +1
        i=i+1
print("end of program")
#Note: it looks like it's getting smaller because the window zooms out
```

Let's try it…why doesn't this look as good – what is wrong?

What are the problems here?

Let's try this!

1. Increase rate to wait less often so it goes faster
2. Decrease position so it's not moving as much
3. Go through loop more times

```
my_sphere=sphere(pos=vector(0,0,0), radius=0.25, color=color.green)
i=1
while(i<=100):
        rate(10)
        my_sphere.pos.x= my_sphere.pos.x +.1
        i=i+1
print("end of program")
```

Give it a name, dx:

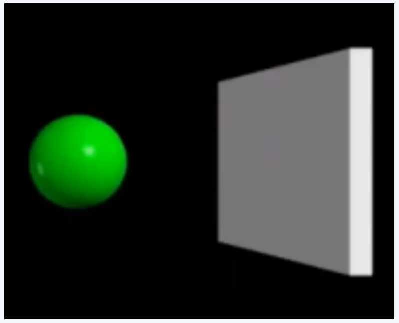
- Use a variable dx (delta x – change in x)

```
my_sphere=sphere(pos=vector(0,0,0), radius=0.25,
color=color.green)
i=1
dx=0.1 #step size
while(i<=100):
        rate(10) #show 1 frame per second
        my_sphere.pos.x= my_sphere.pos.x +dx
        i=i+1
print("end of program")
```

Create Wall for Bouncing!

- What if we want it to come back around if it hits a wall?
- Let’s add a wall. What happened?

```
my_sphere=sphere(pos=vector(0,0,0), radius=0.25, color=color.green)
wall = box(pos=vector(2, 0, 0), size=vector(0.1, 1, 1), color=color.white)
#wall 2 units to the right. Narrow in X, tall in Y, and tall in Z
i=1
dx=0.1
#step size
while(i<=100):
        rate(10)
        my_sphere.pos.x= my_sphere.pos.x +dx
        i=i+1
print("end of program")
```



Check for Wall

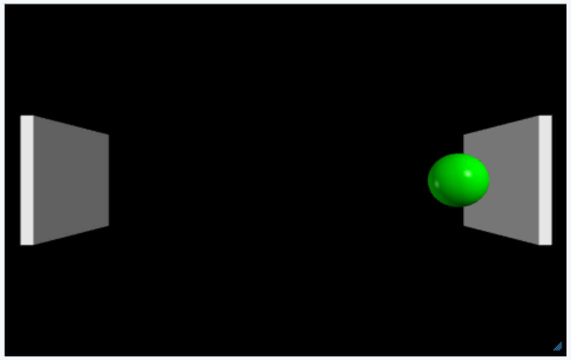
- Logical Tests
  - If the ball moves too far to right, reverse its direction.
- If statement:
  - if(my_sphere.pos.x>=2):
- Then what?
  - Reverse direction
  - dx=-dx

Using If Condition in Animation

```
my_sphere=sphere(pos=vector(0,0,0), radius=0.25, color=color.green)
wall = box(pos=vector(2, 0, 0), size=vector(0.1, 1, 1), color=color.white)
#wall 2 units to the right. Narrow in X, tall in Y, and tall in Z
i=1
dx=0.1
#step size
while(i<=100):
        rate(10)

        #wall is located at x=2 or you could say wall.pos.x
        if(my_sphere.pos.x>=2):
                dx=-dx #start going backwards
        my_sphere.pos.x= my_sphere.pos.x +dx
        i=i+1
print("end of program")
```

How can we improve our Code

- Went into the wall a little
  - Because we are using the number 2 play around with number if you want it to stop before it gets to 2. Remove part of it's size. We won't do this now.
- Kept going left – need something on the other side – we should add another wall…
- We could also add a ceiling and floor, but we haven't kept track of a dy.



A Better way

- We can use Physics.
- In Physics, we have something that can keep track of the speed an object is going in a given direction.

- What is the term for this?

Velocity

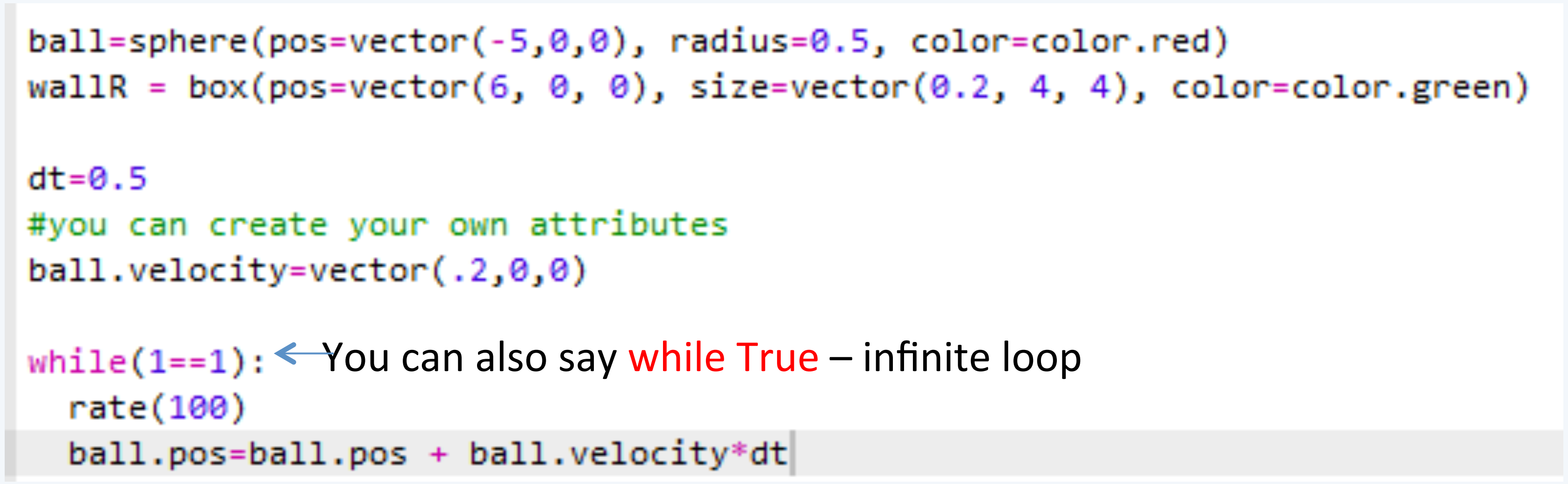
- What is velocity?
  - The speed of something in a given direction.
  - It is a vector quantity

Velocity

- Velocity = displacement(change in position)/ time
  - V= $\triangle x / \triangle t$
  - Δx=new position – old position
- Rewrite to find the new position
  - $\Delta x = v\Delta t$
  - new position - old position = $v\Delta t$
  - new position = old position + $v\Delta t$
- In Vpython
  - **ball.pos = ball.pos + ball.velocity*dt**

We can just use this equation - We will need to specify dt and velocity

Moving Ball – Constant Velocity



```
ball=sphere(pos=vector(-5,0,0), radius=0.5, color=color.red)
wallR = box(pos=vector(6, 0, 0), size=vector(0.2, 4, 4), color=color.green)

dt=0.5
#you can create your own attributes
ball.velocity=vector(.2,0,0)

while(1==1):
  rate(100)
  ball.pos=ball.pos + ball.velocity*dt
```

$\leftarrow$ You can also say while True – infinite loop

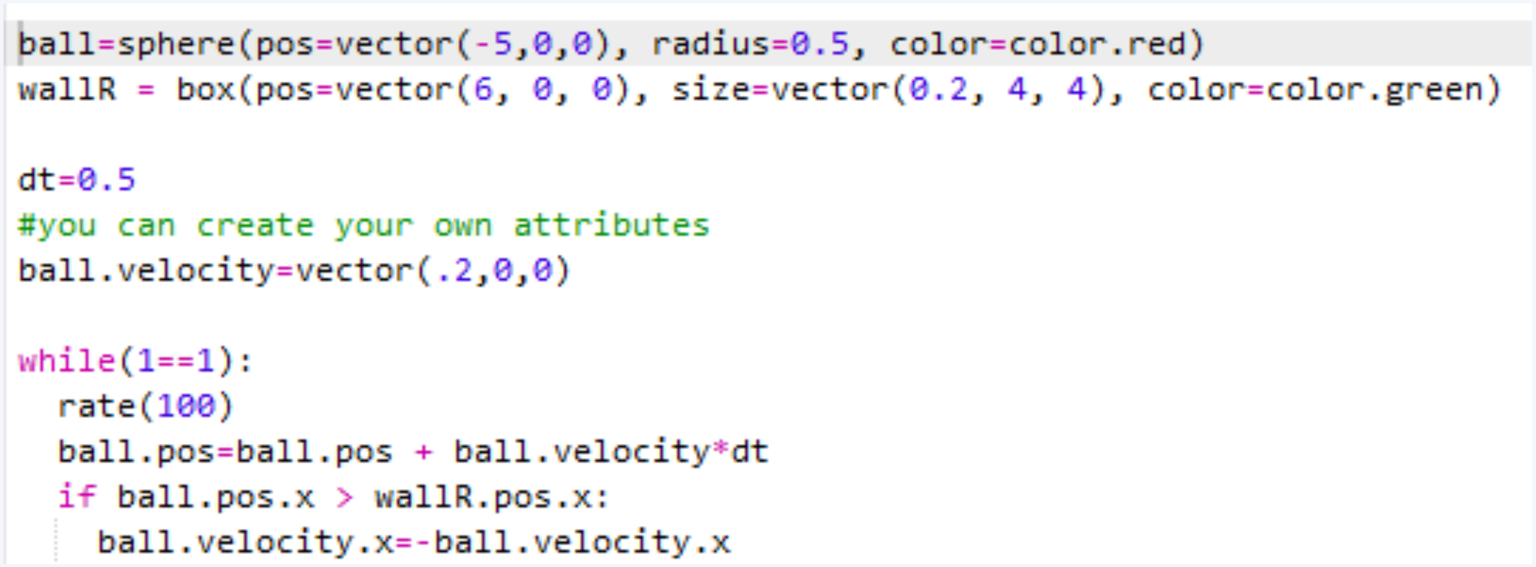
https://www.glowscript.org/docs/VPythonDocs/VPython_Intro.pdf

Making Ball Bounce

- If the ball moves to far to right, reverse its direction.

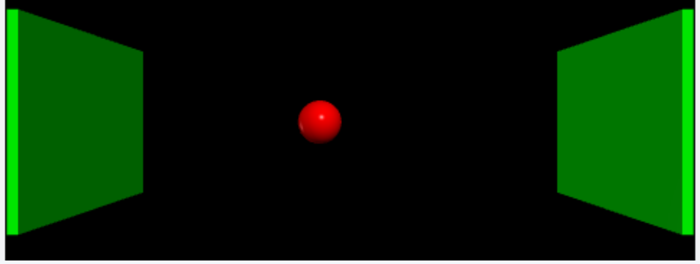
  If ball.x > wallR.x:

  ball.velocity = -ball.velocity



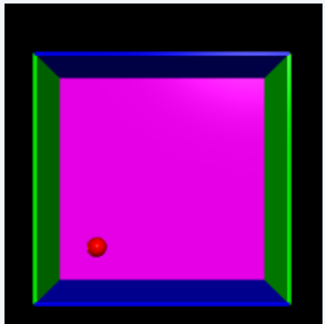
```
ball=sphere(pos=vector(-5,0,0), radius=0.5, color=color.red)
wallR = box(pos=vector(6, 0, 0), size=vector(0.2, 4, 4), color=color.green)

dt=0.5
#you can create your own attributes
ball.velocity=vector(.2,0,0)

while(1==1):
  rate(100)
  ball.pos=ball.pos + ball.velocity*dt
  if ball.pos.x > wallR.pos.x:
    ball.velocity.x=-ball.velocity.x
```

In Class Exercise



- (Full details on handout)
- Create the ball bounce we just did using velocity.
- Add **wallL** on left at (-6,0,0)
- Add a test to have ball bounce off this wall too.

- Extra Time try the following in this order:
  1. Add top and bottom walls and make the ball bounce off it (change y in velocity)
  2. Make the walls touch and form a box.
  3. Add a back wall like a box.
  4. Include an if statement to prevent the ball from coming to the front.
  5. Make the initial velocity this: ball.velocity=vector(.2,.2,.2)



Improve it

- You might have had:

```
If ball.pos.x > wallR.pos.x
        dx=-dx
 If ball.pos.x<wallL.pos.x
        dx=-dx
```

Redundant you could use an OR statement

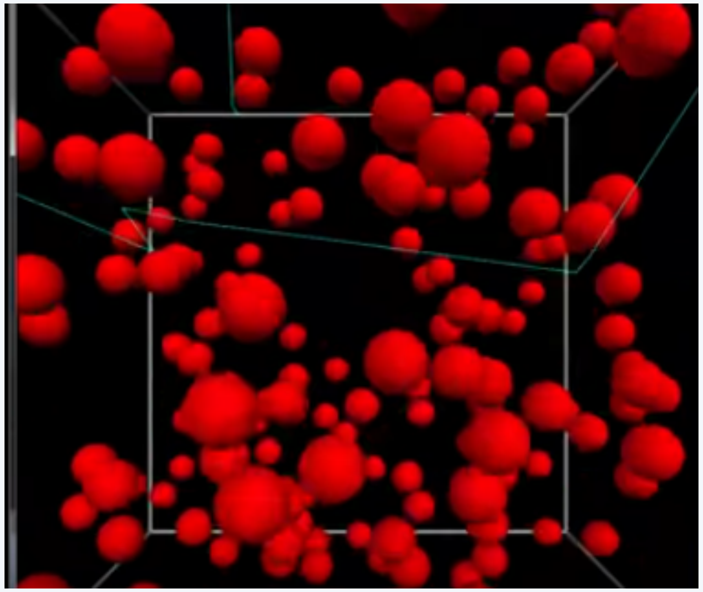
```
if ball.pos.x > wallR.pos.x or ball.pos.x<wallL.pos.x:
                ball.velocity.x=-ball.velocity.x
```

Names

- Creating a few balls and giving them names to refer to it is easy! We did this:

```
ball1 = sphere(pos=vector(0,1,0),color=color.red, radius=0.1)
ball2 = sphere(pos=vector(-2,1,0),color=color.green, radius=0.1)
ball3 = sphere(pos=vector(1,-1.5,0),color=color.yellow, radius=0.1)
```

- But what if we wanted hundreds of balls! What can we do?



Using Lists

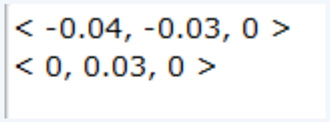
- Can refer to each one by position in list instead of name.
- Create a list:

```
a = sphere(pos=vector(-0.04, -0.03,0),color=color.red, radius=0.005)
b = sphere(pos=vector(0.04,-0.03,0),color=color.red, radius=0.005)
c= sphere(pos=vector(0,0.03,0),color=color.blue, radius=0.005)
particles = [a, b, c]
print(particles[0].pos)
print(particles[2].pos)
```

Programmers start with 0! Print first item's position

Print last element position in list

Numbers in [] called index

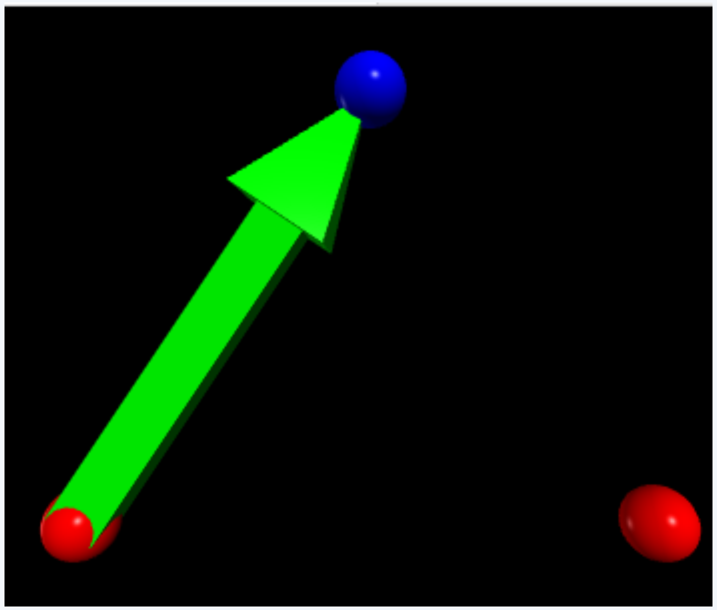


```
< -0.04, -0.03, 0 >
< 0, 0.03, 0 >
```

Arrows

- Add arrows

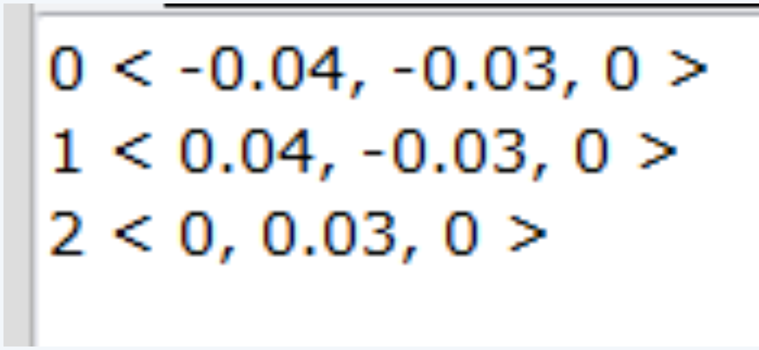
```
arrow(pos=particles[0].pos, axis=particles[2]-particles[0], color=color.green)
```



Looping Through List

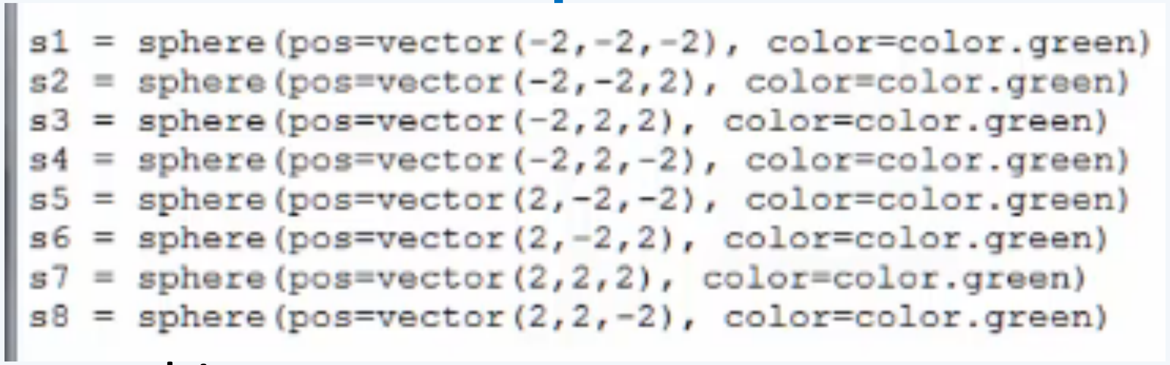
- print(len(particles)) #can tell us there are how many in our list
- 3

```
i=0
while i < len(particles):
        print(i, particles[i].pos)
        i = i+ 1;
```



```
0 < -0.04, -0.03, 0 >
1 < 0.04, -0.03, 0 >
2 < 0, 0.03, 0 >
```

Let’s add 8 Sphere’s into a List

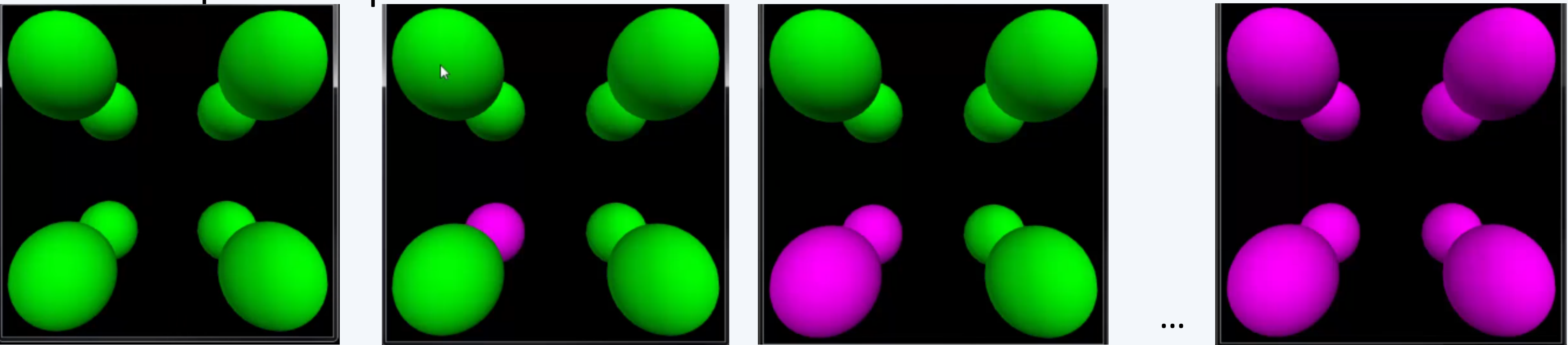


```
s1 = sphere(pos=vector(-2,-2,-2), color=color.green)
s2 = sphere(pos=vector(-2,-2,2), color=color.green)
s3 = sphere(pos=vector(-2,2,2), color=color.green)
s4 = sphere(pos=vector(-2,2,-2), color=color.green)
s5 = sphere(pos=vector(2,-2,-2), color=color.green)
s6 = sphere(pos=vector(2,-2,2), color=color.green)
s7 = sphere(pos=vector(2,2,2), color=color.green)
s8 = sphere(pos=vector(2,2,-2), color=color.green)
```

1. Let’s create this program.
2. Put all 8 spheres in a list.
3. Create a loop to turn the spheres magenta one by one.

Need 3 things in the loop:

   1. Rate statement (use rate 1 to see it change color one by one)
   2. Change color
   3. Update loop index.



Problem

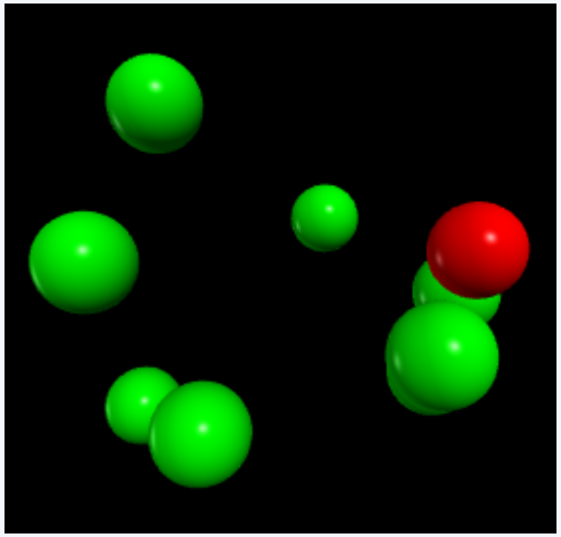
- That wasn't simple!
- That took us a long time to create 8 different spheres and put it in a loop. What if I had 60 spheres?
- We could do it a little differently…

List

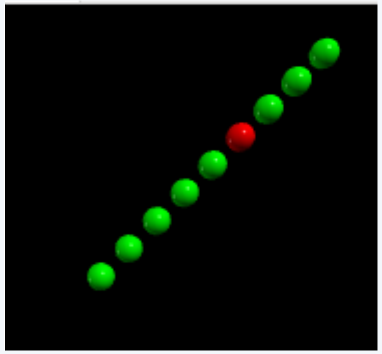
- append – adds an object to the end of a list
- Let’s create objects in random positions and add them to the list in a loop.

Random Balls in a List

```
balls=[]
i=1
while i<10:
        s=sphere(pos=vector.random(), color=color.green, radius=.25)
        balls.append(s)
        i=i+1
balls[5].color=color.red
```



Balls in a Line



```
balls=[]
i=1
dx=.5
dy=.5
while i<10:
        s=sphere(pos=vector(i*dx,i*dy,0), color=color.green, radius=.25)
        balls.append(s)
        i=i+1
balls[5].color=color.red
```

In Class Exercise

- Create a list of at least 10 objects (spheres, boxes, arrows, etc.).
  - You can either create it before the loop OR You can create it inside the loop using append!
- Then try changing either the color or position or some attribute of every item in the list.
  - Use a Loop
  - Use Rate
  - Update Loop Index
- Have fun and be creative!