# Python Classes and Objects

## The Ball Example

# Example: Bouncing Ball

- Lets try to create a bouncing ball class. Essentially this will be a ball that has a velocity and can bounce around a window.
- Specification
  - We want to specify initial position, velocity, color and bounds (where are the walls)
  - We then want to call an update method that moves the ball

# Goal

- Create a Ball class that can display a bouncing ball on the screen
- Ball
  - attributes: color, gravity, airResistance, current location, current velocity
  - method:
    - update – sets the location of the ball to a new location based on time incrementing

# Creating a Ball

- tkinter is Python's standard graphical toolkit.
- canvas is a class that allows drawing things.
  - # (x1,y1) = upper left corner
  - # (x2,y2) = lower right corner
  - myBall = canvas.create_oval(x1, y1, x2, y2, fill="red")

See samplecode/objects/ball/DrawCircle.py

# Screen Layout

# Example: Bouncing Ball

- class Ball:

  def __init__(self, xLoc, yLoc, xVel, yVel, color, leftWall,rightWall, topWall, bottomWall)

  # Should initialize everything

  def update()

  # Should move the ball and let it bounce appropriatly

# Moving something

- Every X seconds, change the location
- **From: http://effbot.org/tkinterbook/canvas.htm**
  - **move(item, dx, dy) #Moves matching items by an offset.**

- myCanvas.move(myBall,5,0) # right 5 pixels

- # Call a function or method after 5 millis
- myCanvas.after(5, someMethod)

- See: MovingCircle.py

# Create the Ball class

- Ball
  - attributes: color, gravity, airResistance, current location, current velocity
- Constructor needs to create a circle on the canvas, and set the appropriate attributes
- See Ball1.py

# Falling Ball

- At time T we are at 100m
- Our velocity is -10m/s
- So, at time t=1 where are we?
- At time t=2 where are we?

- See Ball2.py

# Bouncing Ball

- Everytime we hit the floor, or a wall, just change the direction of our velocity.

- if we're at the floor, start going up
- if we're at the ceiling, start going down
- if we're at the left/right …

- See Ball3.py

# Acceleration

- As the ball bounces, gravity needs to act on it.
- Gravity accelerates at -9.8ms^2. So every second go 9.8m/s faster than the previous second!
- yVelocity = yVelocity + 9.8
- See Ball4.py

# Fix some issues

- Make the “moves” smaller, by dividing all the velocities and times by 10
- Fix the problem that the ball bounces past the bottom of the screen
- Ball5.py
- Ball6.py --- add in lots of balls!

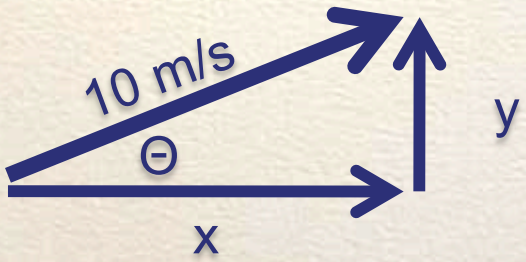
# Design Summary

- Think about each “object” in your system
  - What behaviors should it have?
  - What information does it need to know? What information changes from one instance of this object to the next?
- There are many books on design strategies for object oriented programming!

# Extra Slides

# Bouncing Ball: Physics 101

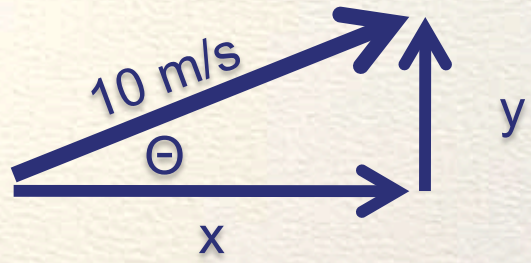
- gravity accelerates items at 9.8m/s$^2$
  - so every second you fall, your speed increases by 9.8m/s
- Our velocity has two components




  - Assuming Θ is 30 degrees
  - cos(Θ) = x / 10
  - sin(Θ) = y / 10

# Bouncing Ball: Physics 101

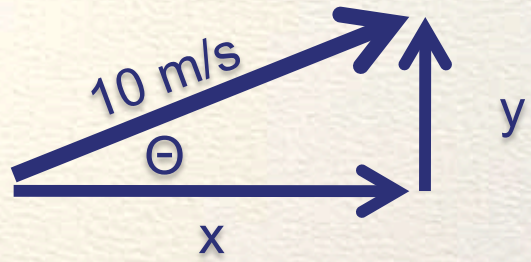
- Our velocity has two components




  - Assuming Θ is 30 degrees
  - cos(Θ) = x / 10
  - sin(Θ) = y / 10
    - x = 10 cos(30) =8.66 m/s
    - y = 10 sin (30) =0.5 m/s

# Bouncing Ball: Physics 101

- Our velocity has two components




  - So, if our ball is travelling at 10 m/s, the y velocity is subject to gravity, but not the x. (we'll ignore wind resistance and all other factors)
  - So the first second we travel 8.66 meters in X and 0.5 meters in Y

# Bouncing Ball: Physics 101

- Our update function will use simulation to keep the ball moving:
  - update():

```
# If we call update every second, then the change in X and Y directions are just their
# velocity (since it's in meters/second)
deltaX = 8.66 # Velocity in X direction never changes
yVelocity = yVelocity – 9.8 # Gravity
deltaY =  yVelocity

# Move the ball
self.canvas.move(self.itm, deltaX, deltaY)
```

This gives us a falling ball, how do we make it bounce?

# Bouncing Ball: Physics 101

- If we hit the “floor”, change the yVelocity from positive to negative, and reduce it some (we bounce a little lower than we started)

```
# Bounce off the "floor"
if self.yLoc > self.bottomWall:
    self.yVelocity = -1 * self.yVelocity * self.bouncyness
    deltaY = self.bottomWall - self.yLoc # Make sure you’re above the floor!
else:
    deltaY = int(self.yVelocity)
self.yLoc += deltaY
```

Now we bounce up and down, what about left and right wall?

# Bouncing Ball: Physics 101

- If we hit the left/right wall, just change our x direction

```
# Bounce off the "wall"
if self.xLoc > self.rightWall or self.xLoc < self.leftWall:
    self.xVelocity *= -1

deltaX = self.xVelocity/5
self.xLoc += deltaX
```

Great… but the balls should stop not keep rolling around

# Bouncing Ball: Physics 101

- If we get to a very small yVelocity, just stop bouncing and rolling.

```
 # The ball isn't bouncing... stop!
if abs(self.yVelocity) < 10 and self.yLoc >= (self.bottomWall-5):
    self.yVelocity = 0
    self.xVelocity = 0
    return
else:
    self.yVelocity += 2 #9.8/5
```

# Bouncing Ball: Physics 101

- Now it's easy to create a whole bunch of balls because they are Objects, and each will maintain it's own state (velocities)

```
for i in range(10):
    rcolor = '#%d%d%d' %(randint(0,9), randint(0,9), randint(0,9)) # Random color
    ball = Ball(randint(left,right), randint(top,bottom), randint(2,20),  \
            randint(2,20), color=rcolor, \
                leftWall=left, rightWall=right, topWall=top, bottomWall=bottom)
    ball.draw(canvas)
    balls.append(ball)
```