Review

- Trigonometry review, SOHCAHTOA
- Coordinate System Transformations
  - Translate
  - Scale
  - Rotate
- Combining/Accumulating Transformations
- resetMatrix(), pushMatrix(), popMatrix()

Notes

- The += operator is a shorthand for + and =

```
i = i + 1     # These pairs of statements have the same effect
i += 1

i = i - 1
i -= 1

s = "a"
s = s + "b"
s += "b"

# When the mouse is pressed, both functions are invoked
onMousePressed += drawEllipse
onMousePressed += drawRect
```

Notes

- The // operator is integer division – no remainder
- The modulus operator (%) returns the remainder of integer division
- If x divides evenly into y, `y % x == 0`

```
d = 11 // 4          # d = 2
r = 11 % 4           # r = 3
```

Physics – Equations of Motion

$$x = x_o + v_o t + \tfrac{1}{2} a t^2$$
$$v = v_o + at$$

$$x_{i+1} = x_i + v_i \Delta t + \tfrac{1}{2} a \Delta t^2$$
$$v_{i+1} = v_i + a \Delta t$$

x = position
v = velocity
a = acceleration
t = time
Δt = time change

If $\Delta t = 1$

$$x_{i+1} = x_i + v_i + \tfrac{1}{2} a$$
$$v_{i+1} = v_i + a$$

A Bouncing Ball

```
# Init all variables
sx = 0.0     # x position
sy = 0.0     # y position
vx = 1.0     # x velocity
vy = 1.0     # y velocity

ay = 0.2     # y acceleration (gravity)
fr = 0.9     # Losses due to friction
```

bounce.py

```
def draw(o, e):
    global sx, sy, vx, vy

    # Equations of Motion
    sx = sx + vx
    sy = sy + vy + 0.5*ay
    vy = vy + ay

    # Bounce off walls
    if sx <= 0.0 or sx >= w:
        vx = -fr*vx

    # Bounce off floor and
    # lose some velocity due to friction
    if sy > (h-10) and vy > 0.0:
        sy = h-10
        vy = -fr*vy

    # Draw at current location
    background(255)
    ellipse(sx, sy, 20, 20)

frameRate(50)
onLoop += draw
loop()
```

bounce.py

Two Bouncing Balls

One approach – duplicate everything

```
# Init all variables
sx = random(0.0, w)           # x position
sy = random(0.0, 10.0)        # y position
vx = random(-3.0, 3.0)        # x velocity
vy = random(0.0, 5.0)         # y velocity

sx2 = random(0.0, w)          # x position
sy2 = random(0.0, 10.0)       # y position
vx2 = random(-3.0, 3.0)       # x velocity
vy2 = random(0.0, 5.0)        # y velocity

ay = 0.2                      # y acceleration (gravity)
fr = 0.9                      # Losses due to friction
```

bounce2.py

```
def draw(o, e):
    global sx, sy, vx, vy, sx2, sy2, vx2, vy2
    background(255)

    # Equations of Motion
    sx = sx + vx
    sy = sy + vy + 0.5*ay
    vy = vy + ay

    sx2 = sx2 + vx2
    sy2 = sy2 + vy2 + 0.5*ay
    vy2 = vy2 + ay

    # Bounce off walls and floor
    if sx <= 0.0 or sx >= w:
        vx = -fr*vx
    if sy >= (h-10.0) and vy > 0.0:
        sy = h-10.0
        vy = -fr*vy

    if sx2 <= 0.0 or sx2 >= w:
        vx2 = -fr*vx2
    if sy2 >= (h-10.0) and vy2 > 0.0:
        sy2 = h-10.0
        vy2 = -fr*vy2

    # Draw ball
    ellipse( sx, sy, 20, 20)
    ellipse( sx2, sy2, 20, 20)
```

bounce2.py

Many Bouncing Balls

```
# Init all variables
sx = []                          # x positions
sy = []                          # y positions
vx = []                          # x velocities
vy = []                          # y velocities

ay = 0.2                         # y acceleration (gravity)
fr = 0.9                         # Losses due to friction

nBalls = 20

# Initialize lists
for i in range(nBalls):
    sx.append( random(0.0, w) )
    sy.append( random(0.0, 10.0) )
    vx.append( random(-3.0, 3.0) )
    vy.append( random(0.0, 5.0) )
```

bounce3.py

```
# Redraw all balls
def draw(o, e):
    global sx, sy, vx, vy
    background(255)

    for i in range(nBalls):
        # Equations of motion
        sx[i] = sx[i] + vx[i]
        sy[i] = sy[i] + vy[i] + 0.5*ay
        vy[i] = vy[i] + ay

        # Bounce off walls and floor
        if sx[i] <= 0.0 or sx[i] >= w:
            vx[i] = -fr*vx[i]
        if sy[i] >= (h-10.0) and vy[i] > 0.0:
            vy[i] = -fr*vy[i]

        # Draw ball
        ellipse( sx[i], sy[i], 20, 20)
```

bounce3.py

Compare bounce1.py and bounce3.py

```
def draw(o, e):
    global sx, sy, vx, vy
    background(255)

    for i in range(nBalls):
        # Equations of motion
        sx[i] = sx[i] + vx[i]
        sy[i] = sy[i] + vy[i] + 0.5*ay
        vy[i] = vy[i] + ay

        # Bounce off walls and floor
        if sx[i] <= 0.0 or sx[i] >= w:
            vx[i] = -fr*vx[i]
        if sy[i] >= (h-10.0) and vy[i] > 0.0:
            sy[i] = h-10
            vy[i] = -fr*vy[i]

        # Draw bouncing ball i
        ellipse( sx[i], sy[i], 20, 20)
```

```
def draw(o, e):
    global sx, sy, vx, vy
    background(255)

    # Equations of Motion
    sx = sx + vx
    sy = sy + vy + 0.5*ay
    vy = vy + ay

    # Bounce off walls and floor
    if sx <= 0.0 or sx >= w:
        vx = -fr*vx
    if sy > (h-10) and vy > 0:
        sy = h-10
        vy = -fr*vy

    # Draw bouncing ball
    ellipse(sx, sy, 20, 20)
```

bounce1.py vs. bounce3.py

Our four lists might look like this…

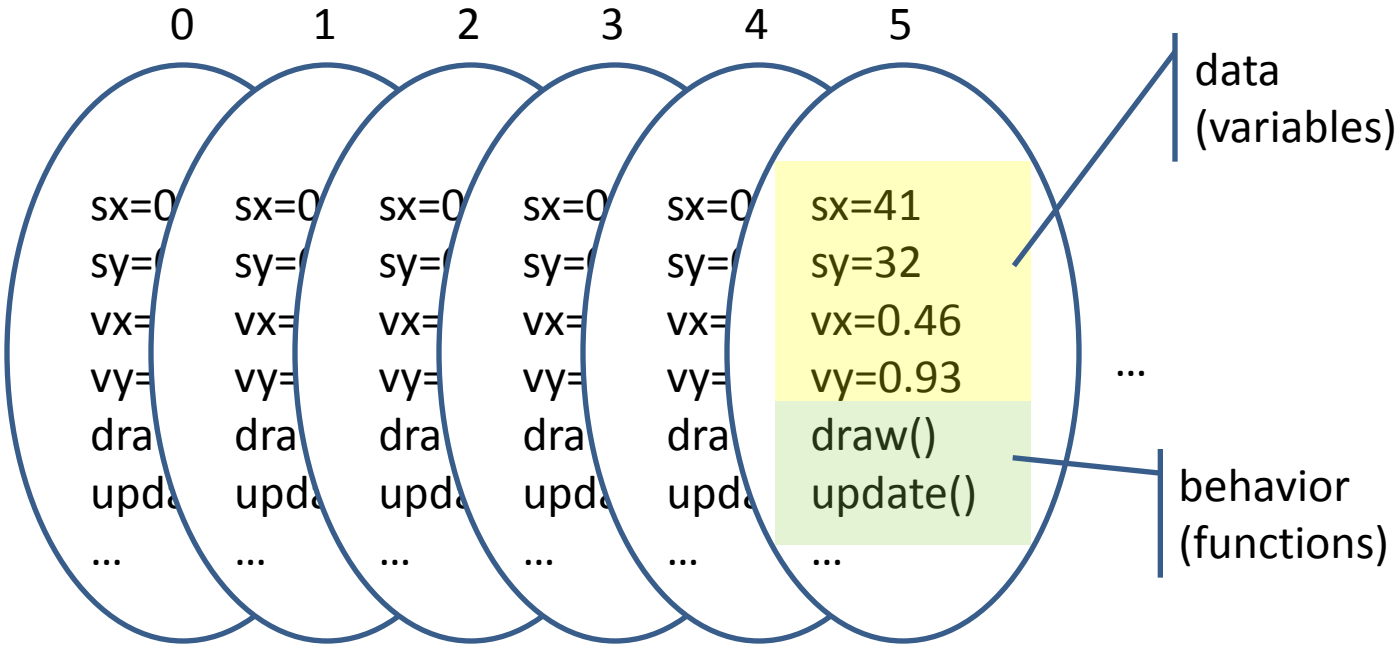
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
sx	41	68	49	3	24	5	2	38	53	72	58	11	68	82	68	28	8	5	29	42	11
sy	32	73	81	61	32	68	37	4	18	19	5	98	75	08	.6	49	23	58	65	68	63
vx	0.46	0.85	0.99	0.25	0.61	0.78	0.74	0.2	0.85	0.7	0.66	0.39	0.99	0.15	0.11	0.85	0.18	0.15	0.64	0.61	0.82
vy	0.93	0.67	0.1	0.67	0.22	0.05	0.37	0.89	0.22	0.86	0.96	0.93	0.7	0.73	0.27	0.98	0.04	0.36	0.66	0.15	0.37

Our four lists might look like this…

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
sx	41	68	49	3	24	5	2	38	53	72	58	11	68	82	68	28	8	5	29	42	11
sy	32	73	81	61	32	68	37	4	18	19	5	98	75	08	.6	49	23	58	65	68	63
vx	0.46	0.85	0.99	0.25	0.61	0.78	0.74	0.2	0.85	0.7	0.66	0.39	0.99	0.15	0.11	0.85	0.18	0.15	0.64	0.61	0.82
vy	0.93	0.67	0.1	0.67	0.22	0.05	0.37	0.89	0.22	0.86	0.96	0.93	0.7	0.73	0.27	0.98	0.04	0.36	0.66	0.15	0.37

But we think of them like this … all data items for the same ball

Stored like this …



For each ball …
… we want the **<u>data</u>** called **instance variables**,
… as well as the **<u>behavior</u>** called **instance methods** (functions)
… to be grouped together into a single software unit with which we can work

OBJECTS

The Python Class Statement

```
class BBall:

    # Constructor
    def __init__(self, arg1, arg2):

        # Init instance variables
        self.sx = random(0.0, w)
        self.sy = random(0.0, 10.0)
        self.vx = random(-3.0, 3.0)
        self.vy = random(0.0, 5.0)

    # Define methods
    def draw(self):
        pass

    # Another method
    def update(self):
        pass
```

Constructor: __init__(self)

- A special method in the class statement named __init__()
  - First argument is a reference to the new instance
- By convention, first argument is named 'self'
  - Establish/init instance variables in body of __init__()
- Assign variables using self and dot-notation
- Perform any other initialization necessary

```
class BBall:
    # The constructor chooses a random location and velocity
    # and sets the given diameter and color
    def __init__(self, diam, clr):
        self.sx = random(0.0, w)
        self.sy = random(0.0, 10.0)
        self.vx = random(-3.0, 3.0)
        self.vy = random(0.0, 5.0)
        self.diameter = diam
        self.clr = clr
```

Creating Instances

- Invoke the class as if it was a function
  - This invokes the __init__() constructor, passing arguments provided
  - The first argument (self) is inserted automatically

```
# Set number of BBall objects and init list
nBalls = 10
bballs = []

# Create all Ball instances and add to balls list
for i in range(nBalls):
    diam = random(10, 30)
    clr = color( random(255), random(255), random(255) )
    bball = BBall( diam, clr )
    bballs.append( bball )
```

Methods

- Functions defined within a class statement
- ‘Owned’ by instances
- First argument, ‘self’, is automatically inserted
- Use ‘self’ as reference to the current instance

```
class BBall:
    ...
    # Update the position and velocity of this Ball instance
    def update(self):
        # Equations of Motion
        self.sx = self.sx + self.vx
        self.sy = self.sy + self.vy + + 0.5*ay
        self.vy = self.vy + ay

        # Bounce off walls and floor
        if self.sx <= 0.0 or self.sx >= w:
            self.vx = -fr*self.vx
        if self.sy >= (h-0.5*self.diameter) and self.vy > 0.0:
            self.vy = -fr*self.vy

    def draw(self):
        fill( self.clr )
        ellipse( self.sx, self.sy, self.diameter, self.diameter )
```

Methods

- Invoked using an instance and dot-notation
- First argument, ‘self’, is automatically inserted

```
# Function that updates and draws balls
def draw(o, e):
    background(255)

    # Update and redraw all Ball instances
    for i in range(nBalls):
        bballs[i].update()
        bballs[i].draw()

# Handle loop event and start looping
frameRate(50)
onLoop += draw
startLoop()
```

```
# Declare a BBall class
class BBall:
    # The constructor chooses a random starting location and velocity
    # and sets the give diameter and color
    def __init__(self):
        self.sx = random(0.0, width())
        self.sy = random(0.0, 10.0)
        self.vx = random(-3.0, 3.0)
        self.vy = random(0.0, 5.0)

    # Update the position and velocity of this instance
    def update(self):
        # Equations of Motion
        self.sx = self.sx + self.vx
        self.sy = self.sy + self.vy + + 0.5*ay
        self.vy = self.vy + ay

        # Bounce off walls and floor
        if self.sx <= 0.0 or self.sx >= w:
            self.vx = -fr*self.vx
        if self.sy >= (h-10) and self.vy > 0.0:
            self.vy = -fr*self.vy

    def draw(self):
        fill( 255, 0, 0 )
        ellipse( self.sx, self.sy, 20, 20 )
```

bounce4.py

```
# Set the number of BBall objects and init the list that holds all
nBalls = 10
bballs = []

# Create all Ball instances and add to balls list
for i in range(nBalls):
    bball = BBall()
    bballs.append( bball )

# Function that updates and draws balls
def draw(o, e):
    background(255)

    # Update and redraw all Ball instances
    for i in range(nBalls):
        bballs[i].update()
        bballs[i].draw()

# Handle loop event and start looping
frameRate(50)
onLoop += draw
startLoop()
```

bounce4.py

Expand the BBall Class

```
# Declare a BBall class that encapsulates all attributes and methods of a Ball
class BBall:
    # The constructor chooses a random starting location and velocity
    # and sets the given diameter and color
    def __init__(self, diam, clr):
        self.sx = random(0.0, w)
        self.sy = random(0.0, 10.0)
        self.vx = random(-3.0, 3.0)
        self.vy = random(0.0, 5.0)
        self.diameter = diam
        self.clr = clr

    # Update the position and velocity of this Ball instance
    def update(self):
        # Equations of Motion
        self.sx = self.sx + self.vx
        self.sy = self.sy + self.vy + + 0.5*ay
        self.vy = self.vy + ay

        # Bounce off walls and floor
        if self.sx <= 0.0 or self.sx >= w:
            self.vx = -fr*self.vx
        if self.sy >= (h-0.5*self.diameter) and self.vy > 0.0:
            self.vy = -fr*self.vy

    def draw(self):
        fill( self.clr )
        ellipse( self.sx, self.sy, self.diameter, self.diameter )
```

bounce5.py