Python features, *motivated* by VPython…

Python features, *motivated* by VPython...

Tuples are similar to lists, but they're parenthesized:

```
T = (4,2)      x = (1,0,0)
```

example of a two-element ***tuple*** named T and a three-element tuple named x

not vectors!

```
def f(x=3, y=17):
    return 10*x + y
```

examples of ***default and named inputs*** in a function definition

Tuples!

Lists that use parentheses are called ***tuples***:

```
T = ( 4, 2 )

T
(4, 2)

T[0]
4


T[0] = 42
Error!


T = ('a',2,'z')
```

Tuples are immutable lists: you can't change their elements...

...but you can always redefine the whole variable, if you want!

+ Tuples are more memory + time efficient
+ Tuples ***can*** be dictionary keys: ***lists can't be***
- ***But you can't change tuples' elements...***

Default – *and named* – inputs!

Functions can have ***default input values*** and can take ***named inputs***

```
def f(x=3, y=17):
    return 10*x + y
```

example of an ordinary function call – totally OK

```
f(4,2)
```


example of ***default inputs***

```
f()
```


example using only one ***default input***

```
f(1)
```


example of a ***named input***

```
f(y=1)
```

Named inputs

```
def f(x=2, y=11):
    return x + 3*y
```

Input your name(s) = _____

`f(3,1)` ➡

`f()` ➡

`f(3)` ➡

`f(y=4,x=2)` ➡

What will these function calls to `f` return?

None of the above are 42!

but they all share a factor with it! - Eli B. '17

What call to `f` returns the string **'Lalalalala'** ?

These are tuples – they work like lists!

What is `f( ( ), (1,0) )` ?

What is the ***shortest*** call to `f` returning 42?

it's only four characters, too!

Extra... what does this return? **y = 60; x = -6; f(y=x,x=y)**

Named inputs

```
def f(x=2, y=11):
    return x + 3*y
```

Try this on the back page first...

`f(3,1)` → **6**

`f()` → **35**

`f(3)` → **36**

`f(y=4,x=2)` →

What will these function calls to `f` return?

None of the above are 42!

but they all share a factor with it! - Eli B. '17

What call to `f` returns the string **'Lalalalala'**? → `f('Lala','la')`

These are tuples – they work like lists!

What is `f( ( ), (1,0) )`? → `(1,0,1,0,1,0)`

What is the ***shortest*** call to `f` returning 42?

it's only four characters, too!

Extra... what does this return? **y = 60; x = -6; f(y=x,x=y)** →

VPython ~ GlowScript!

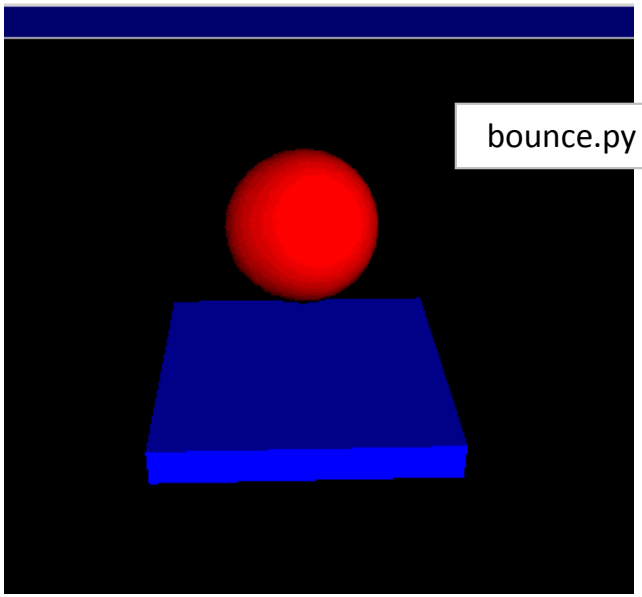
www.glowscript.org/

Where were we... ?

stonehenge.py

built *by* and *for* physicists to simplify 3d simulations

lots of available classes, objects and methods in its **API**



bounce.py

Using GlowScript / vPython...

glowscript.org

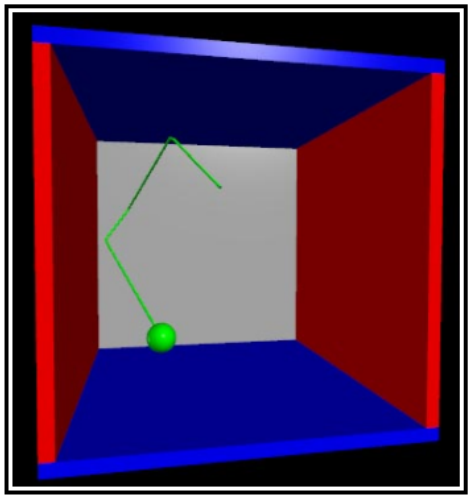
Signed in as **zdodds**(**Sign out**)
Help

GlowScript is an easy-to-use, powerful environment for creating 3D animations and publishing them on the web. Here at glowscript.org, you can write and run GlowScript programs right in your browser, store them in the cloud for free, and easily share them with others. Thanks to the **RapydScript-NG** compiler, you can use VPython here.

New version 2.6: Can resize a canvas; new graph titles

The Help provides full documentation.

You are signed in as **zdodds** and your programs are **here**.
Your files will be saved here, but it is a good idea to keep your own copies of any important files.



examples...

GlowScript 2.6
Example programs | **Forum**

Documentation

documentation...

This is for later on. (The documentation links for the browser-based

- http://www.glowscript.org/ - home page, where you login and access your programs
- http://www.glowscript.org/docs/GlowScriptDocs/index.html - docs for each object, the
- http://www.glowscript.org/#/user/GlowScriptDemos/folder/Examples/ - examples (you

API

... stands for *Application Programming Interface*

a ***programming*** description of how to use a software library

A demo of vPython's API:

```
# the simplest possible vpython program:
box( color = vector(1, 0.5, 0) )

# try changing the color: the components are
#   red, green, blue    each from 0.0 to 1.0

# then, add a second parameter:   size=vector(2.0,1.0,0.1)
#       the order of those three #s: Length, Height, Width

# then, a third parameter:  axis=vector(2,5,1)
#       the order of those three #s: x, y, z
```

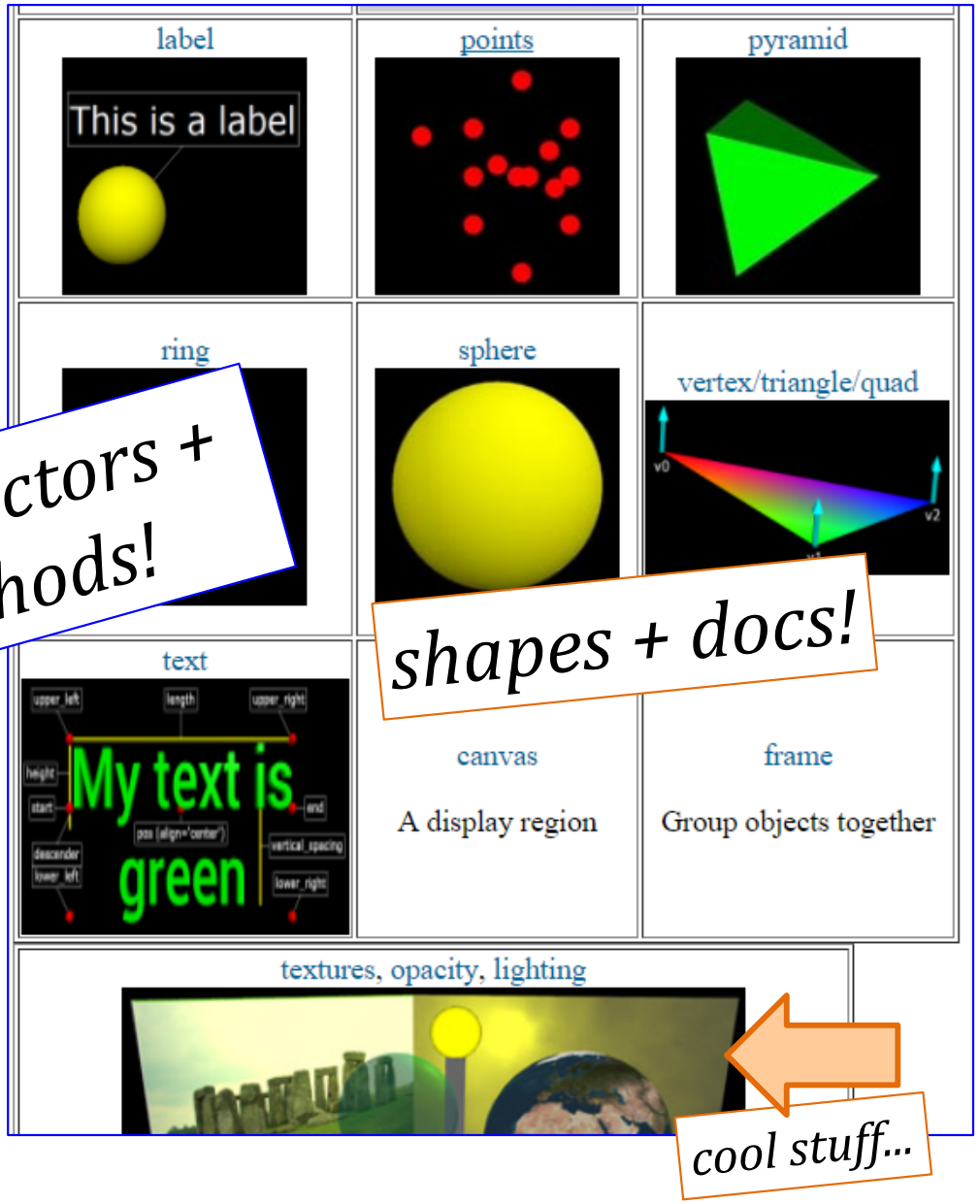
What's **box**?
What's **color**?
What's **vector**?
Getting used to everything!

vPython example API call(s)

I'm hAPI about APIs!

API

... stands for *Application Programming Interface*

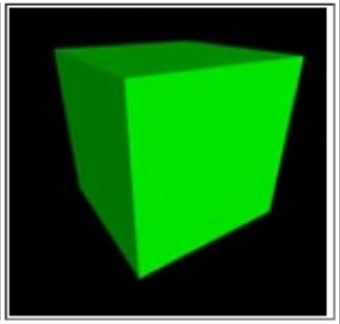


API

... stands for *Application Programming Interface*

constructor + default arguments; data!

box



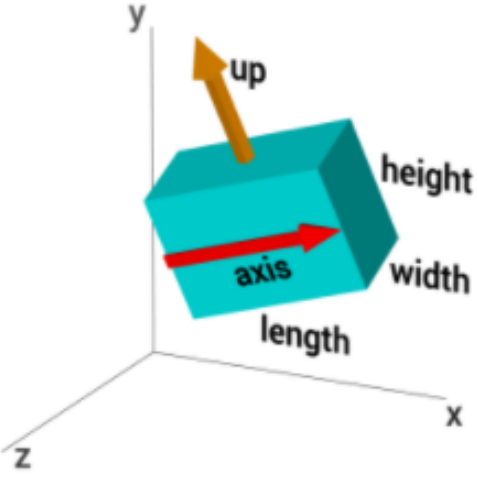
IHere is how to create a box object:

```
mybox = box(   pos=vec(x0,y0,z0),
  size=vec(L,H,W) )
```

The given position is in the center of the box, at (x0, y0, z0). This is different from cylinder, whose pos attribute is at one end of the cylinder. Just as with a cylinder, we can refer to the individual vector components of the box as mybox.pos.x, mybox.pos.y, and mybox.pos.z. For this box, we have mybox.axis = vec(1, 0, 0). Note that the axis of a box is just like the axis of a cylinder.

For a box that isn't aligned with the coordinate axes, additional issues come into play. The orientation of the length of the box is given by the axis:

```
mybox = box(
 pos=vec(x0,y0,z0),
 axis=vec(a,b,c),
 size=vec(L,H,W) )
```



The axis attribute gives a direction for the length of the box, and the length, height, and width of the box are given as before.

You can rotate the box around its own axis by changing which way is "up" for the box, by specifying an up attribute for the box that is different from the up vector of the

vectors

b.pos, b.vel,... are vectors

```
b.vel = vector(1,0,0)
```

vel.x vel.y vel.z

named components

```
b.pos = vector(0,0,0)
```

scalar multiplication

```
b.pos = b.pos + b.vel*0.2
```

component-by-component addition

let's compare with tuples...

vectors

act like "*arrows*"

The vector Object

The vector object is not a displayable object but is a powerful aid to 3D computations.

vector(x,y,z)

Returns a vector object with the given components, which are made to be floating-point (that is, 3 is converted to 3.0).

Vectors can be added or subtracted from each other, or multiplied by an ordinary number. For example,

```
v1 = vector(1,2,3)
v2 = vector(10,20,30)
print(v1+v2) # displays <1 22 33>
print(2*v1)  # displays <2 4 6>
```

You can refer to individual components of a vector:

v2.x is 10, v2.y is 20, v2.z is 30

It is okay to make a vector from a vector: vector(v2) is still vector(10,20,30).

The form vector(10,12) is shorthand for vector(10,12,0).

A vector is a Python sequence, so v2.x is the same as v2[0], v2.y is the same as v2[1], and v2.z is the same as v2[2].

Vector functions

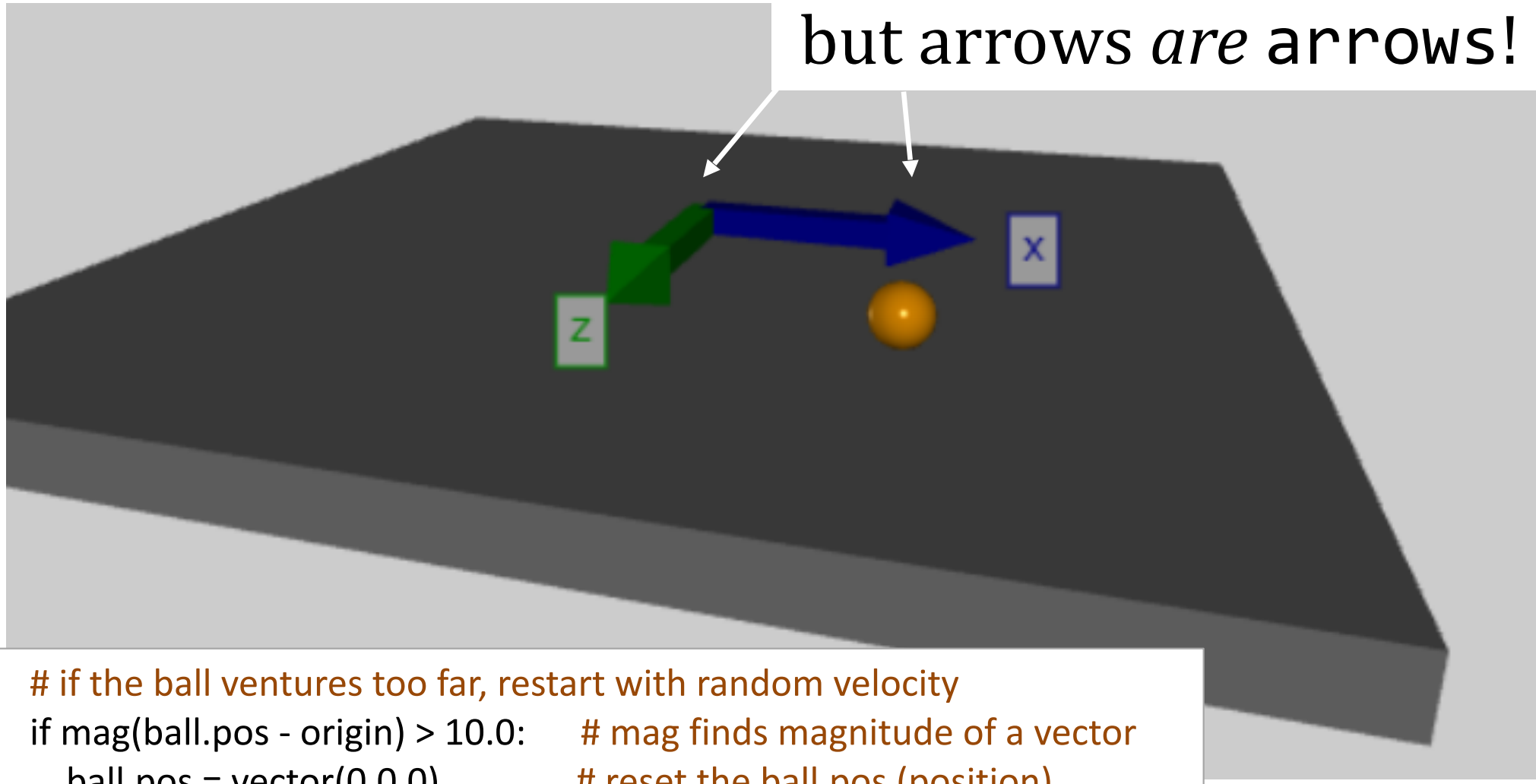
The following functions are available for working with vectors:

www.glowscript.org/docs/GlowScriptDocs/vector.html

vectors

act like "*arrows*"

but arrows *are* arrows!




```
# if the ball ventures too far, restart with random velocity
if mag(ball.pos - origin) > 10.0:      # mag finds magnitude of a vector
    ball.pos = vector(0,0,0)           # reset the ball.pos (position)
    ball.vel = 4.2*vector.random()     # set a random velocity
    ball.vel.y = 0.0                   # with no y component (no vertical)
    print("velocity is now:", ball.vel)
```

vectors!

lots of support…
(don't write your own)

Vector functions

The following functions are available for working with vectors:

mag(A) = A.mag = |A|, the magnitude of a vector

mag2(A) = A.mag2 = |A|*|A|, the vector's magnitude squared

norm(A) = A.norm() = A/|A|, a unit vector in the direction of the vector

hat(A) = A.hat = A/|A|, a unit vector in the direction of the vector; an alternative to A.norm(), based on the fact that unit vectors are customarily written in the form **ĉ**, with a "hat" over the vector

dot(A,B) = A.dot(B) = A dot B, the scalar dot product between two vectors

cross(A,B) = A.cross(B), the vector cross product between two vectors

diff_angle(A,B) = A.diff_angle(B), the angle between two vectors, in radians

proj(A,B) = A.proj(B) = dot(A,norm(B))*norm(B), the vector projection of A along B

comp(A,B) = A.comp(B) = dot(A,norm(B)), the scalar projection of A along B

A.equals(B) is True if **A** and **B** have the same components (which means that they have the same magnitude and the same direction)

vec.random() produces a vector each of whose components are random numbers in the range -1 to +1

www.glowscript.org/docs/GlowScriptDocs/index.html

Documentation!

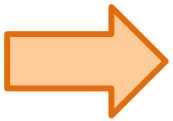
Rotating a vector

There is a function for rotating a vector:

```
v2 = rotate(v1, angle=a, axis=vec(x,y,z))
```

The angle must be in radians. The default axis is (0,0,1), for a rotation counterclockwise in the xy plane around the z axis. There is no origin for rotating a vector. You can also write v2 = v1.rotate(angle=theta, axis=vec(1,1,1)). There is also a rotate capability for objects.

The JavaScript versions are v2 = rotate(v1, {angle:a, axis=vec(x,y,z}) and v2 = v1.rotate({angle:a, axis=vec(x,y,z}).

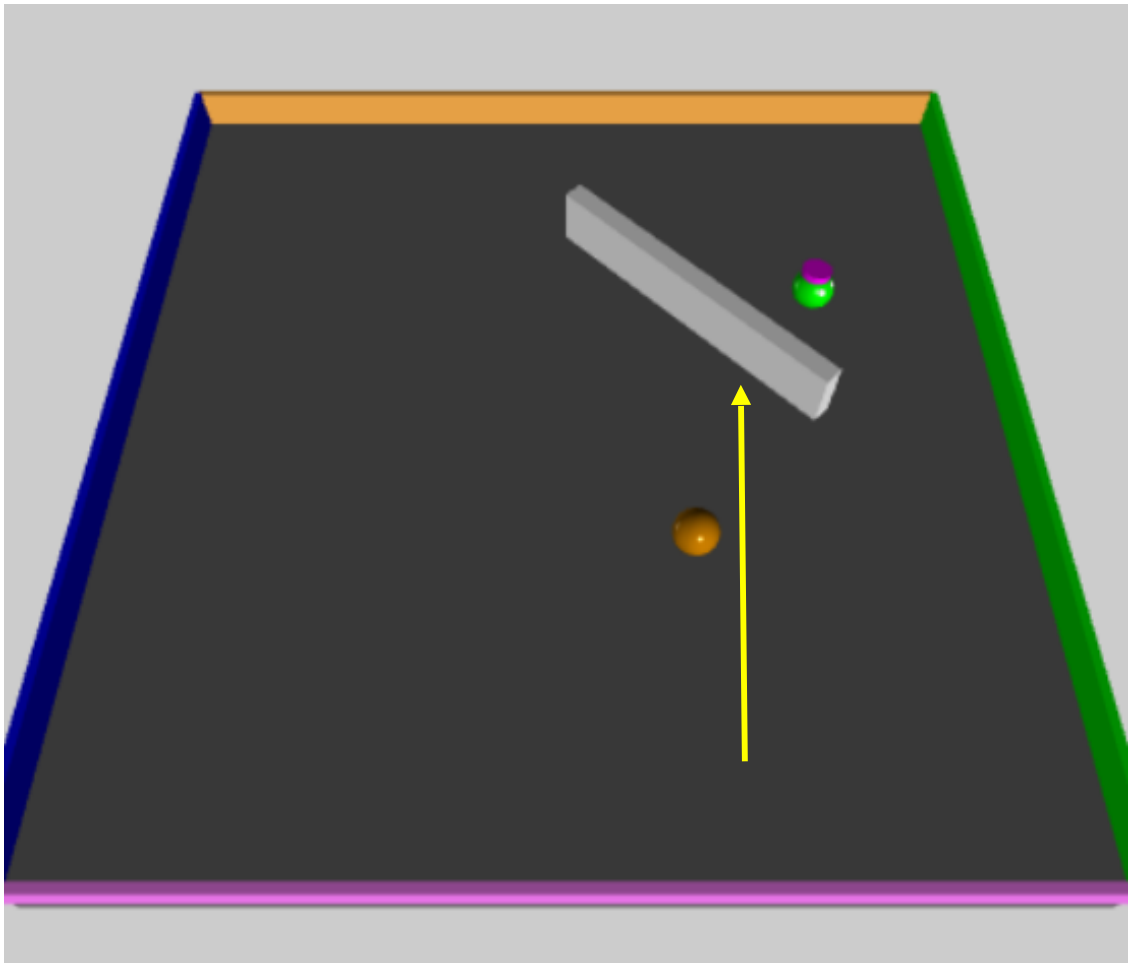


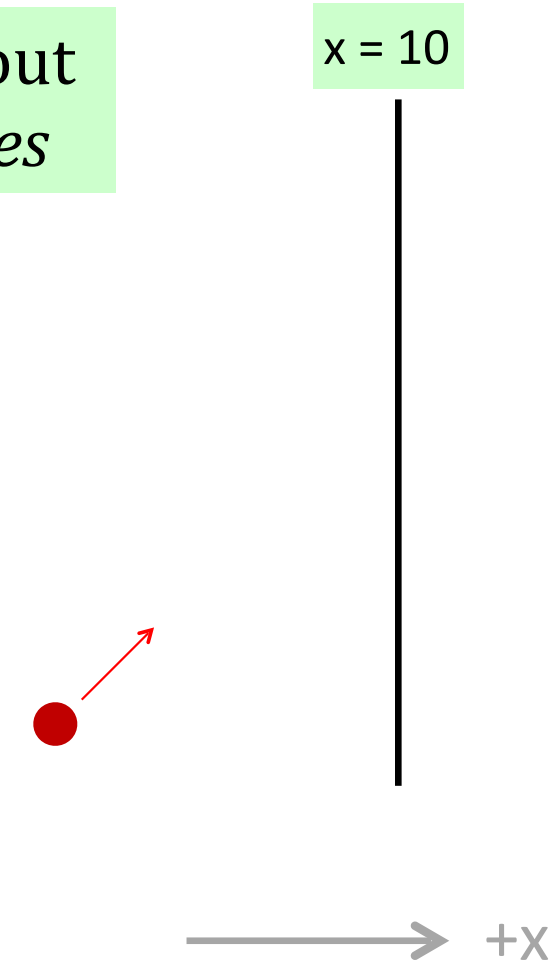
There are functions for converting between degrees and radians, where there are 2*pi radians in 360 degrees:

```
radians(360)  is equivalent to 2*pi
degrees(2*pi) is equivalent to 360
```

vPython: Linear + Spherical collisions...

At least some of the game needs to be about *detecting collisions* and *changing velocities*

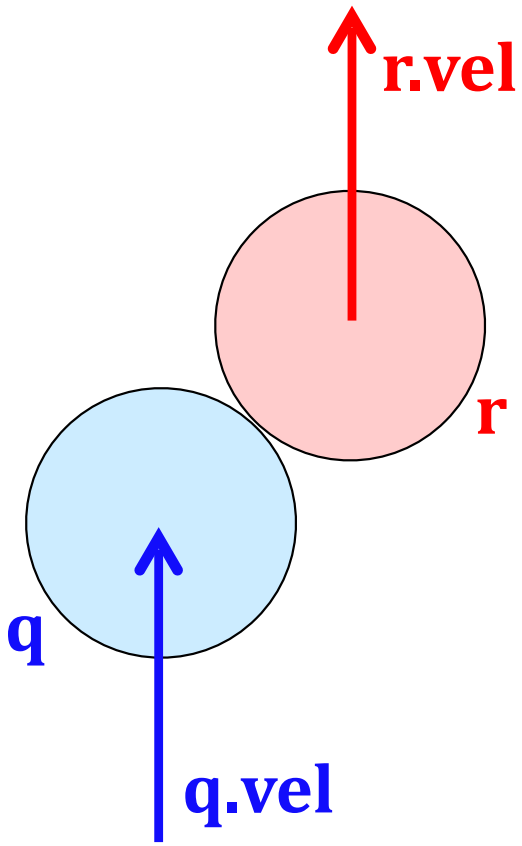




How to **bounce**?

What else to do?

Spherical collisions



0 Zeroth approximation:

Stop **q**. *Undo any overlap.*

Make **r**.vel = **q**.vel.

1 First approximation:

Stop **q**. *Undo any overlap.*

Compute **d** = **r**.pos – **q**.pos

Make **r**.vel = **d**

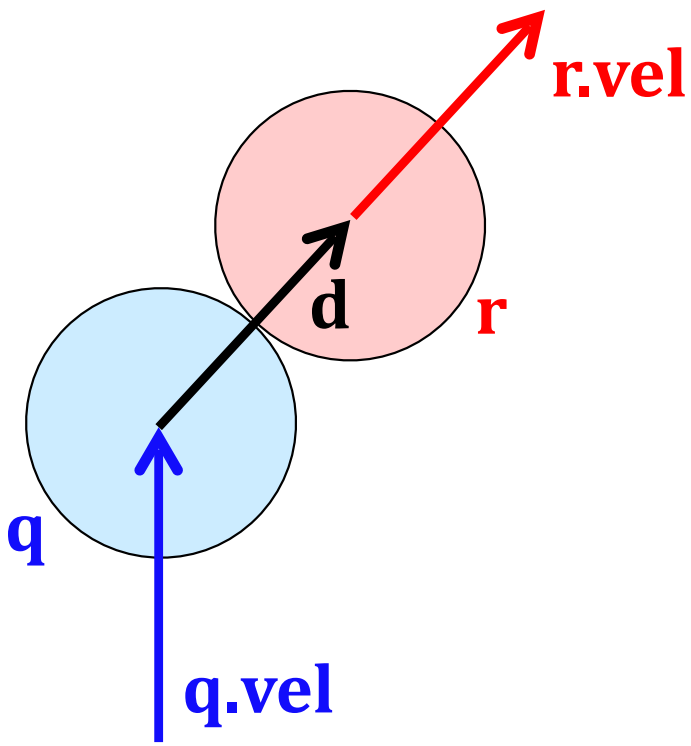
2 Second approximation:

Same as **first**, but

Make **q**.vel = $\mathbf{d}^{\perp}$, at 90° from **d**

Reality is just three eyes away!

Spherical collisions



0 Zeroth approximation:

Stop **q**. *Undo any overlap.*

Make **r**.vel = **q**.vel.

1 First approximation:

Stop **q**. *Undo any overlap.*

Compute **d** = **r**.pos – **q**.pos

Make **r**.vel = **d**

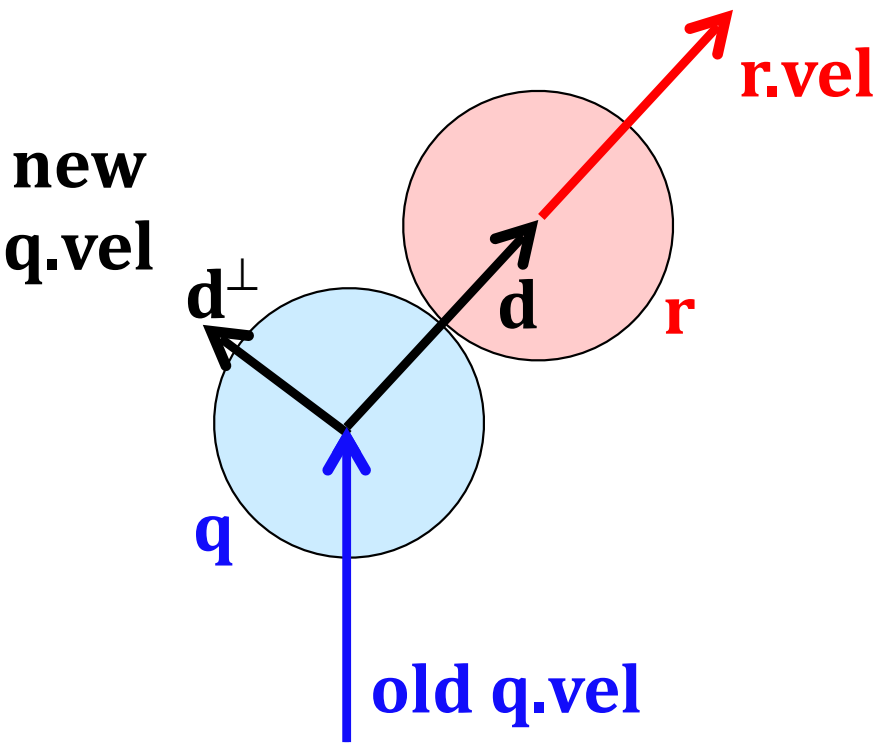
2 Second approximation:

Same as **first**, but

Make **q**.vel = $\mathbf{d}^{\perp}$, at 90° from **d**

Reality is just two eyes away!

Spherical collisions



0 Zeroth approximation:

Stop **q**. *Undo any overlap.*

Make **r**.vel = **q**.vel.

1 First approximation:

Stop **q**. *Undo any overlap.*

Compute **d** = **r**.pos – **q**.pos

Make **r**.vel = **d**

2 Second approximation:

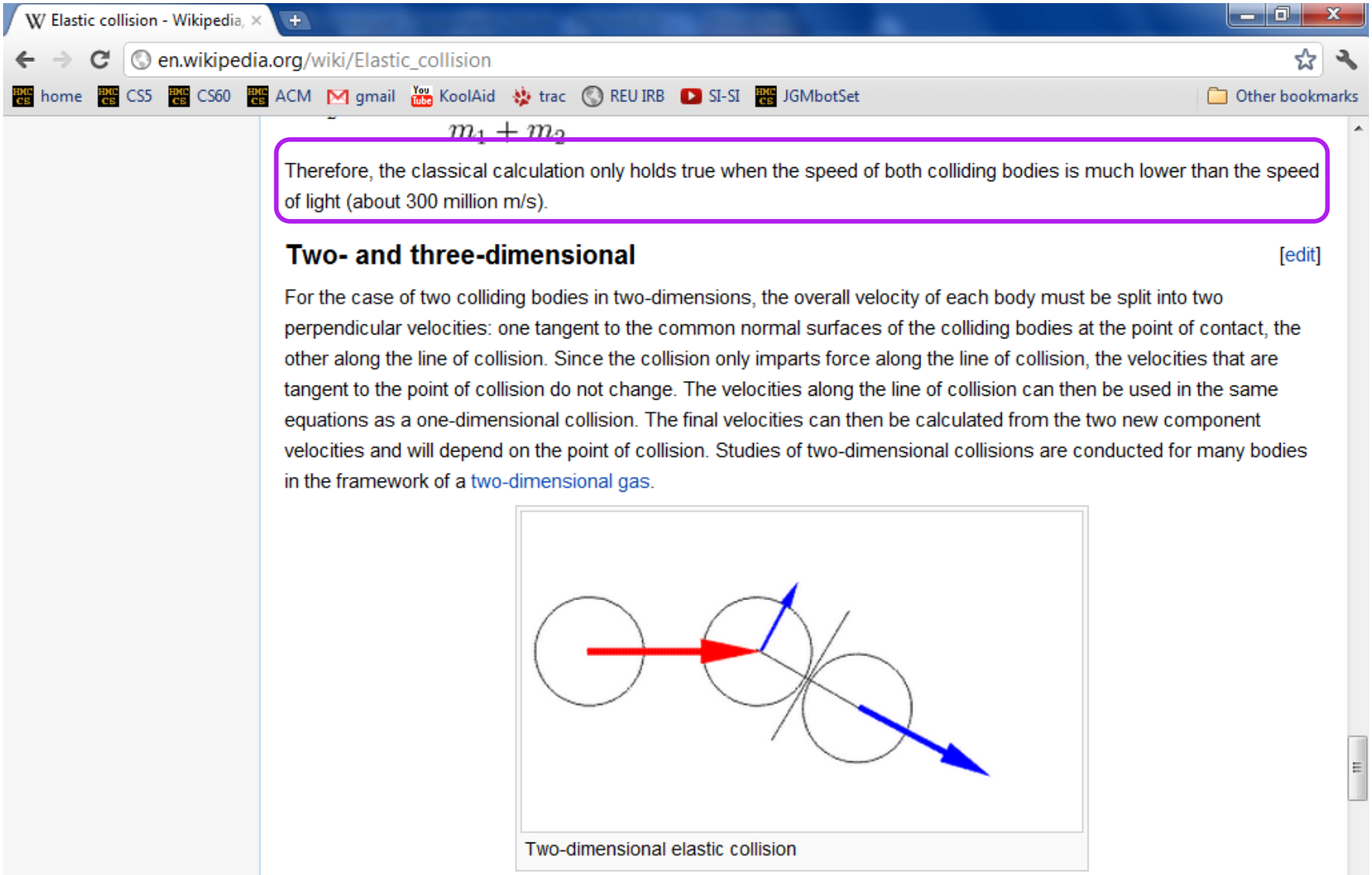
Same as **first**, but

Make **q**.vel = $\mathbf{d}^{\perp}$, at 90° from **d**

Reality is just one eye away!

vPool – physics?

http://en.wikipedia.org/wiki/Elastic_collision



equations below...

hw11pr1 goals

(0) Try out VPython: Get your bearings (*axes!*)

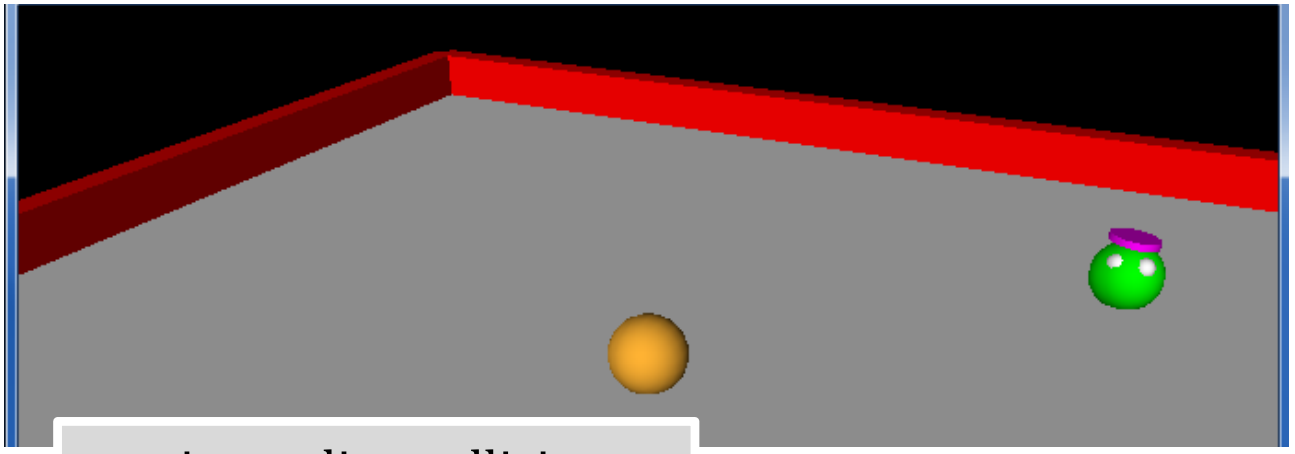
(1) Make guided changes to the starter code...

(2) Expand your ***walls*** and ***wall-collisions***…

(3) Improve your interaction/game!

(4) *Optional*: add scoring, enemies, or a moving target, hoops, traps, holes, etc.

Collisions…



point-to-line collisions

```
# if the ball hits wallA
if ball.pos.z < wallA.pos.z:      # hit - check for z
    ball.pos.z = wallA.pos.z       # bring back into bounds
    ball.vel.z *= -1.0                    # reverse the z velocity

# if the ball hits wallB
if ball.pos.x < wallB.pos.x:       # hit - check for x
    ball.pos.x = wallB.pos.x        # bring back into bounds
    ball.vel.x *= -1.0                      # reverse the x velocity

# if the ball collides with the alien, give a vertical velocity
if mag( ball.pos - alien.pos ) < 1.0:
    print("To infinity and beyond!")
    alien.vel = vector(0,1,0)
```

point-to-point collisions

compound

Home Pictures of 3D objects

Choose a 3D object ▾ | Work with 3D objects ▾ | Canvases/Events ▾

compound

The **compound** object lets you group objects together and manage them as though they were one object, by specifying in the usual way pos, color, size (and length, width, height), axis, up, opacity, shininess, emissive, and texture. Moreover, the display of a complicated compound object is faster than displaying the individual objects one at a time. (In GlowScript version 2.1 the **details were somewhat different**.)

The object shown above is a compound of a cylinder and a box:

```
handle = cylinder( size=vec(1,.2,.2),
                   color=vec(0.72,0.42,0) )

head = box( size=vec(.2,.6,.2),
            pos=vec(1.1,0,0),
            color=color.gray(.6) )

hammer = compound([handle, head])
hammer.axis = vec(1,1,0)
```

The size of the object: After creating the compound named "hammer", **hammer.size** represents the size of the bounding box of the object.

compound

Home Pictures of 3D objects

Choose a 3D object ▾ | Work with 3D objects ▾ | Canvases/Events ▾

The **compound** object lets you group objects together and manage them as though they were one object, by specifying in the usual way pos, color, size (and length, width, height), axis, up, opacity, shininess, emissive, and texture. Moreover, the display of a complicated compound object is faster than displaying the individual objects one at a time. (In GlowScript version 2.1 the **details were somewhat different**.)

The object shown above is a compound of a cylinder and a box:

What's what here?

```
alien_body = sphere( size=1.0*vector(1,1,1), pos=vector(0,0,0), color=color.green )
alien_eye1 = sphere( size=0.3*vector(1,1,1), pos=.42*vector(.7,.5,.2), color=color.white )
alien_eye2 = sphere( size=0.3*vector(1,1,1), pos=.42*vector(.2,.5,.7), color=color.white )
alien_hat = cylinder( pos=0.42*vector(0,.9,-.2), axis=vector(.02,.2,-.02),
                                                 size=vector(0.2,0.7,0.7), color=color.magenta)
alien_objects = [alien_body, alien_eye1, alien_eye2, alien_hat]

com_alien = compound( alien_objects, pos=starting_position )
```

```
# +++ start of EVENT_HANDLING section - separate functions for
#                       keypresses and mouse clicks...

def keydown_fun(event):
    """ function called with each key pressed """
    ball.color = randcolor()
    key = chr(event.which)
    ri = randint( 0, 10 )
    print("key:", key, ri)  # prints the key pressed - caps only...

    amt = 0.42   # "strength" of the keypress's velocity changes
    if key in 'WI&': # all capitals!
        ball.vel = ball.vel + vector(0,0,-amt)
    if key in 'A%J':
        ball.vel = ball.vel + vector(-amt,0,0)
    if key in 'S(K':
        ball.vel = ball.vel + vector(0,0,amt)
    if key in "D'L":
        ball.vel = ball.vel + vector(amt,0,0)
    if key in " ":
        ball.vel = vector(0,0,0) # reset! via the spacebar
        ball.pos = vector(0,0,0)
```

random change of the sphere's color

printing is great for debugging!

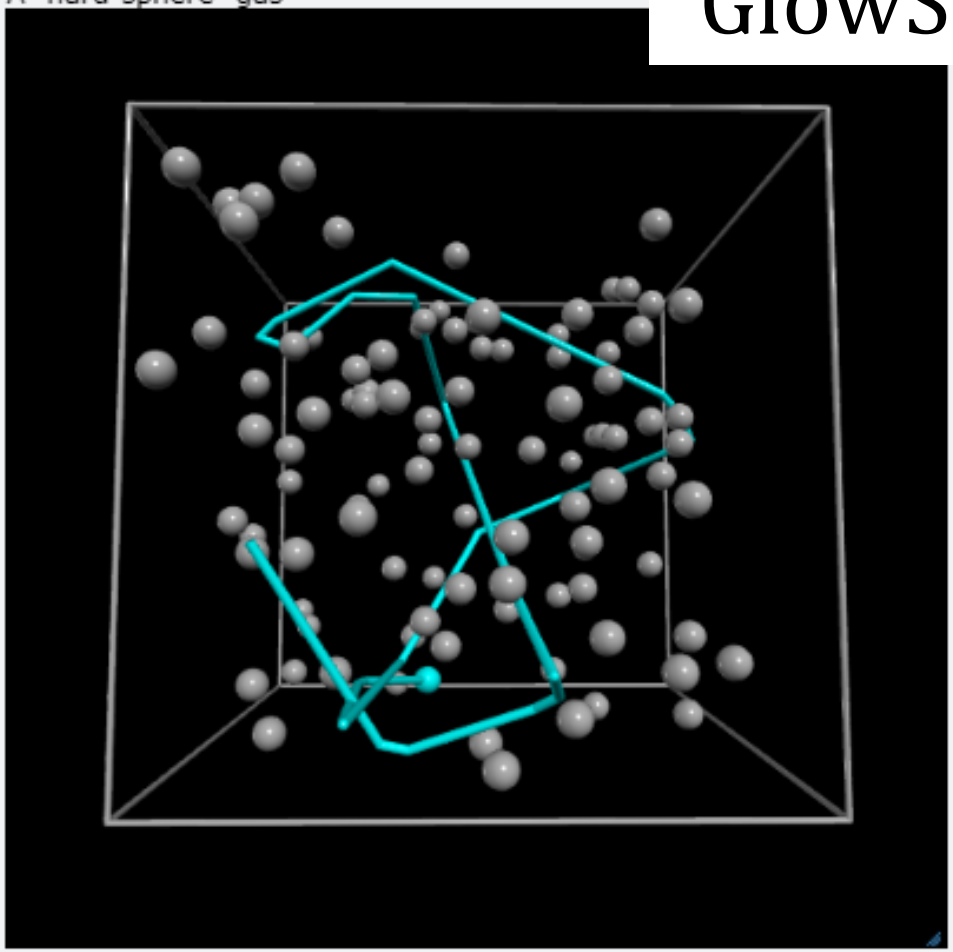
variables make it easy to change behavior across many lines of code
(here, all four motion directions)

have shortcuts to make your game easier -- or reset it!

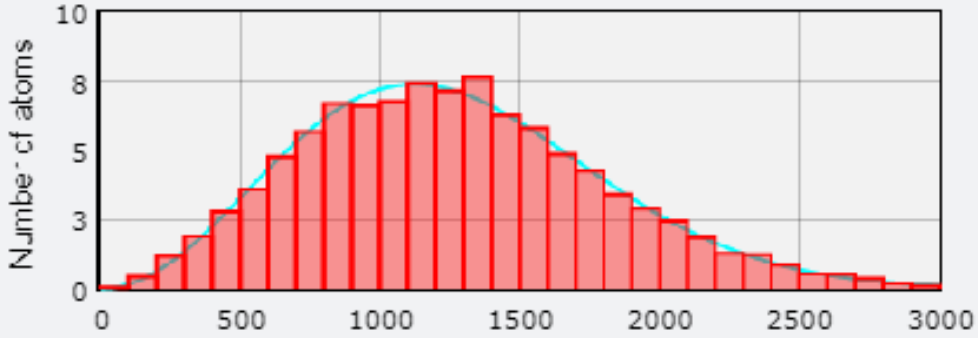
key presses...

GlowScript / vPython examples…

A "hard-sphere" gas

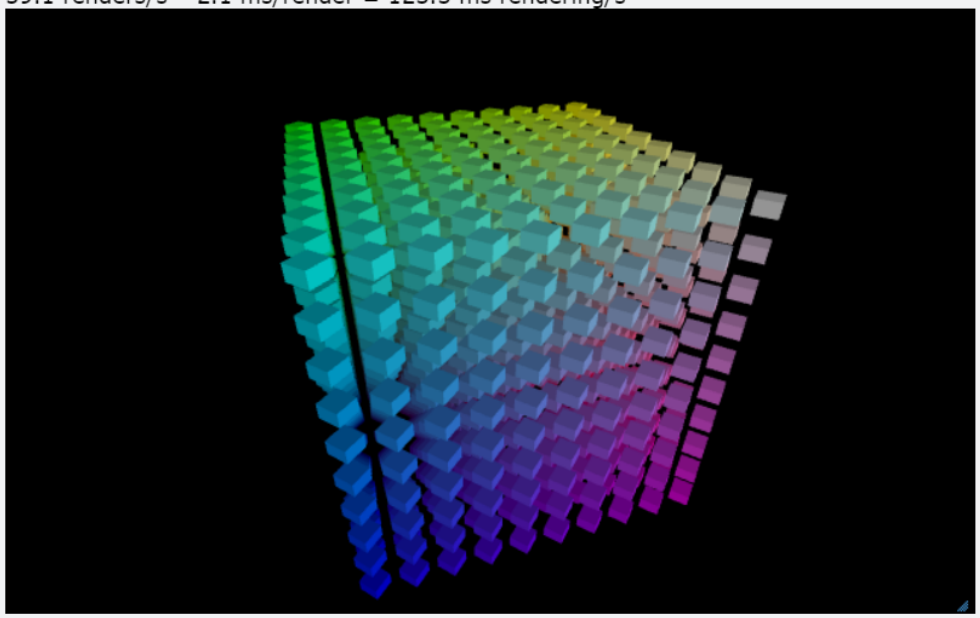


Theoretical and averaged speed distributions (meters/sec).
Initially all atoms have the same speed, but collisions change the speeds of the colliding atoms. One of the atoms is marked and leaves a trail so you can follow its path.



10 by 10 by 10= 1000 rotating cubes

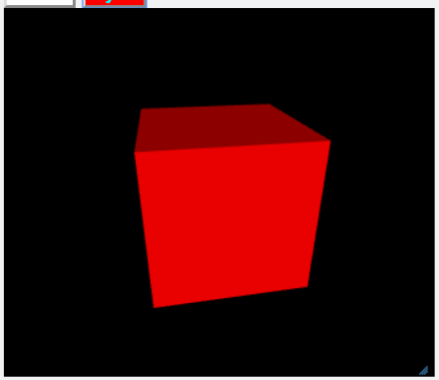
59.1 renders/s * 2.1 ms/render = 123.3 ms rendering/s



Click a box to turn it white

Widgets (buttons, etc.)

Pause Cyan



Vary the rotation speed:

1.50 radians/s

Cyan Choose an object

Red Transparent

Hey! I see what's happening here!