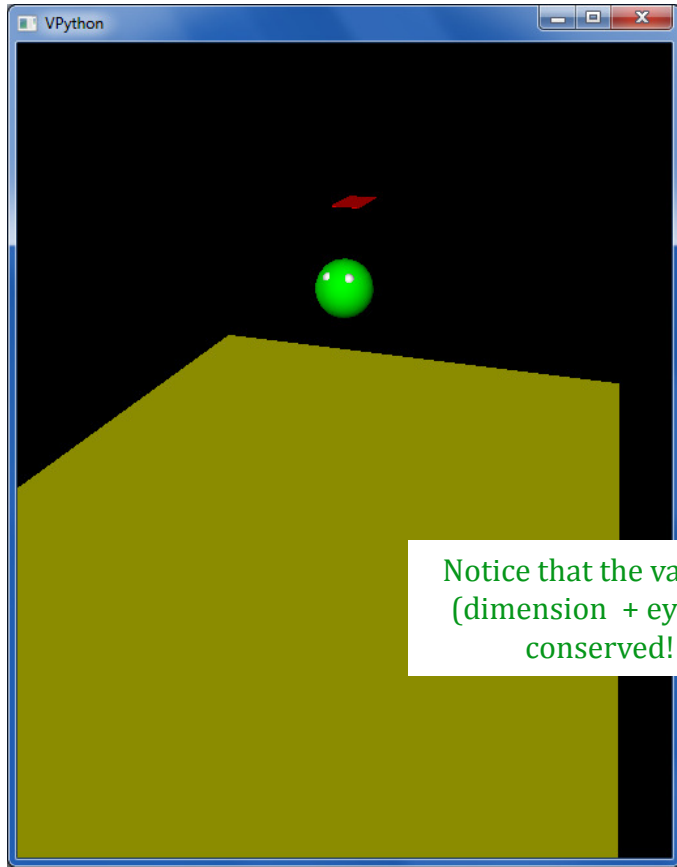
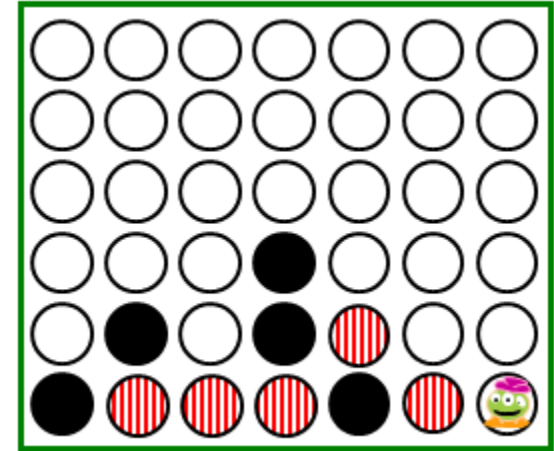


# This week's classes

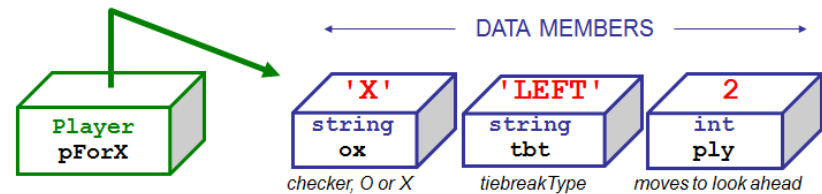


**Alien**  
playground

Homework #11,  
due 11/25



What data does a computer AI player need?



**Connect 4**  
**Player**

Opening Remarks

## The Hidden Technology That Makes Twitter Huge

By Paul Ford | November 07, 2013

## Bloomberg Businessweek Technology

Global  
Economics

Companies &  
Industries

Politics & Policy

Technology

Markets &  
Finance

Innovation &  
Design

Lifestyle



257k



# Twitter's power?

# a public API

# Object-oriented philosophy...

Consider the tweet. It's short—140 characters and done—but hardly simple. If you open one up and look inside, you'll see a remarkable clockwork, with 31 publicly documented data fields. Why do these tweets, typically born of a stray impulse, need to carry all this data with them?

While a tweet thrives in its timeline, among the other tweets, it's also designed to stand on its own, forever. Any tweet might show up embedded inside a million different websites. It may be called up and re-displayed years after posting. For all their supposed ephemerality, tweets have real staying power.

Once born, they're alone and must find their own way to the world, like a just-hatched sea turtle crawling to the surf. Luckily they have all of the information they need in order to make it: A tweet knows the identity of its creator, whether bot or human, as well as the location from which it originated, the date and time it went out, and dozens of other little things—so that wherever it finds itself, the tweet can be reconstituted. Millennia from now an intelligence coming across a single tweet could, like an archaeologist pondering a chunk of ancient skull, deduce an entire culture.



Illustration by David Parkins

[Behind this week's cover](#)

# Python objects used in VPython...

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

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

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

# 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
- ***But you can't change tuples' elements...***

# Tuple problems...

Creating 1-tuples would  
seem like a problem!



A bug from yesterday's **Board** class:

```
W = 4
s = " ",
for col in range(W):
    s += str(col), " "
```

yields a surprising result for **s**:

# Tuple problems...

Creating 1-tuples would  
seem like a problem!



A bug from yesterday's **Board** class:

```
W = 4
s = " ",
for col in range(W):
    s += str(col), " "
```

yields a surprising result for **s**:

```
(' ', '0', ' ', '1', ' ', '2', ' ', '3', ' ')
```

# Python details used in VPython...

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

```
def f(x=9, y=33):  
    return x + y
```

example of default input  
values for x and y

# Python details used in VPython...

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

```
def f(x=9, y=33):  
    return x + y
```

example of default input  
values for x and y

`f()`

`f(1)`

example of a  
*named input*

`f(y=1)`



```
def f(x=2, y=0):  
    return x*(1+4*y)
```

# Named inputs

Input your name(s) = \_\_\_\_\_

What will these function calls to **f** return?

**f** (3, 2)

**f** (3)

**f** ()

**f** (y=3, x=1)

What is *a* call to **f** that returns 42?

What is the *shortest* call to **f** returning 42?  
fewest # of characters

Extra!

feedback *feedback*

# feedback *feedback*

What is something you'd *keep* about CS5 ...?

CS5! Picobot.

What is something you'd *change about / get rid of / add to* CS5 ...?

PICOBOT

What is something you'd *change about / get rid of / add to* CS5 ...?

I didn't like Picobot, and I would  
not be sad to see it go

# feedback *feedback*

What is something you'd *keep* about CS5 ...?

CS5! Picobot.

What is something you'd *change about* / *get rid of* / *add to* CS5 ...?

PICOBOT

What is something you'd *change about* / *get rid of* / *add to* CS5 ...?

I didn't like Picobot, and I would  
not be sad to see it go

Something you'd *keep* about CS5 ...?

Recursion + all the recursion practice

/ *get rid of* / *add to* CS5 ...?

→ RECURSION! ☹️ (it's so hard)

# feedback *feedback*

Something you'd *keep* about CS5 ...?

I really enjoyed logism

Something you'd *keep* about CS5 ...?

I really like all the material we're studying, especially logism

Something you'd *change about / get rid of /*

Circuits :(

Something you'd *change about / get rid of /*

No more Logism!

u'd *change about / get rid of / add*

More projects, less logicism.

# feedback *feedback*

I particularly liked Logisim &  
Hmmm.  
..?

Something you'd *change about* / *get rid of* /  
I would like to get rid of HMMM, ~~change the~~

Something you'd *keep* about CS5 ...?

More music &

Something you'd *change about* / *get rid of* / *add to* CS5 ...?

→ get rid of sound editing (star wars tracks)

# feedback ?

Something you'd *keep* about CS5 ...?

I don't find the hardware stuff not that interesting

Something you'd *change about / get rid of / add to* CS5 ...?

I don't really like the reading, but don't get rid of it.

Something you'd *change about / get rid of / add to* CS5 ...?

Hmmmm, nothing

Something you'd *change about / ge*

More candy.

Other thoughts optional, but 142% welcome:

Remember to carry your towel

# feedback ?

Something you'd *keep* about CS5 ...?

I don't find the hardware stuff not that interesting

Something you'd *change about* / *get rid of* / *add to* CS5 ...?

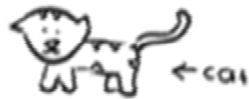
Something you really like the reading, but don't get rid of it.

Something you'd *change about* / *get rid of* / *add to* CS5 ...?

The project portion of CS at the end of the semester

Something you

Other thoughts optional, but 142% welcome:



← cat



← penguin



← cookie

Something you

More candy.

Other thoughts optional, but

Remember to carry your



# feedback *feedback*

**4.51hrs.**

1.9 st dev.

**2.90**

0.90 st dev.

**3.03**

0.40 st dev.

On average how much time  
per week do you spend on  
CS5 *outside class + lab*?

How does CS5's workload  
compare to other classes  
you're taking this term?

How would you judge  
the *pace* of CS5?

# feedback *feedback*

4.51hrs.

1.9 st dev.

2.90

0.90 st dev.

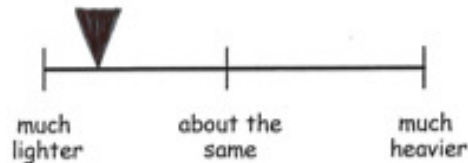
3.03

0.40 st dev.

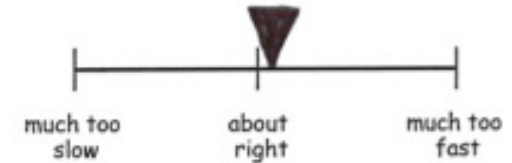
On average how much time  
per week do you spend on  
CS5 *outside class + lab*?

1.5

How does CS5's workload  
compare to other classes  
you're taking this term?



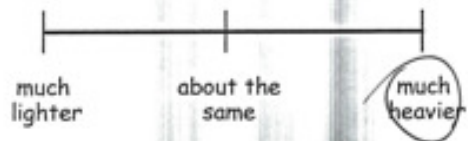
How would you judge  
the *pace* of CS5?



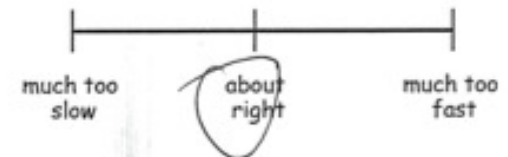
On average how much time  
per week do you spend on  
CS5 *outside class + lab*?

6

How does CS5's workload  
compare to other classes  
you're taking this term?



How would you judge  
the *pace* of CS5?



# feedback *feedback*

4.51hrs.

1.9 st dev.

2.90

0.90 st dev.

3.03

0.40 st dev.

The image shows a student's handwritten feedback on a survey form. The form has three questions, each with a horizontal line for an answer. The first question is 'On average how much time per week do you spend on CS5 outside class + lab?'. The second question is 'How does CS5's workload compare to other classes you're taking this term?'. The third question is 'How would you judge the *pace* of CS5?'. The student has written '6' on the first line, 'More pop tarts' on the second line, and 'much heavier' on the third line. The third line also has a scale with labels: 'much lighter', 'about the same', 'much heavier', 'much too slow', 'about right', and 'much too fast'. The 'much heavier' and 'about right' labels are circled.

On average how much time per week do you spend on CS5 outside class + lab?

How does CS5's workload compare to other classes you're taking this term?

How would you judge the *pace* of CS5?

6

much lighter about the same much heavier much too slow about right much too fast

feedback *feedback*

end with libraries

# VPython

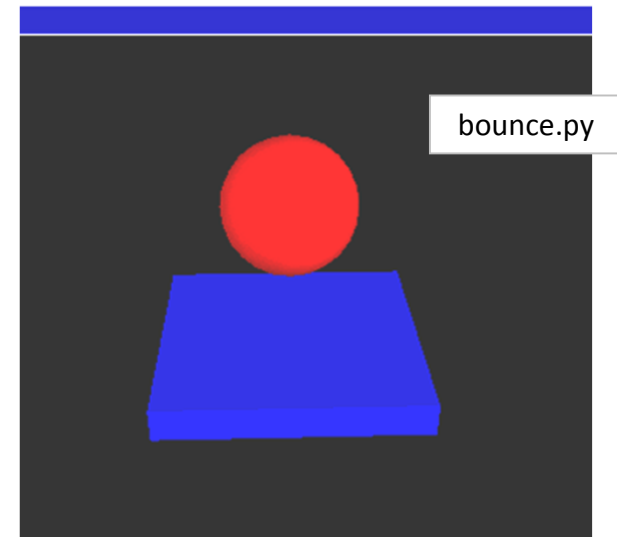
www.vpython.org

**VPython**  
3D Programming for  
Ordinary Mortals



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

lots of available  
classes, objects and  
methods in its API



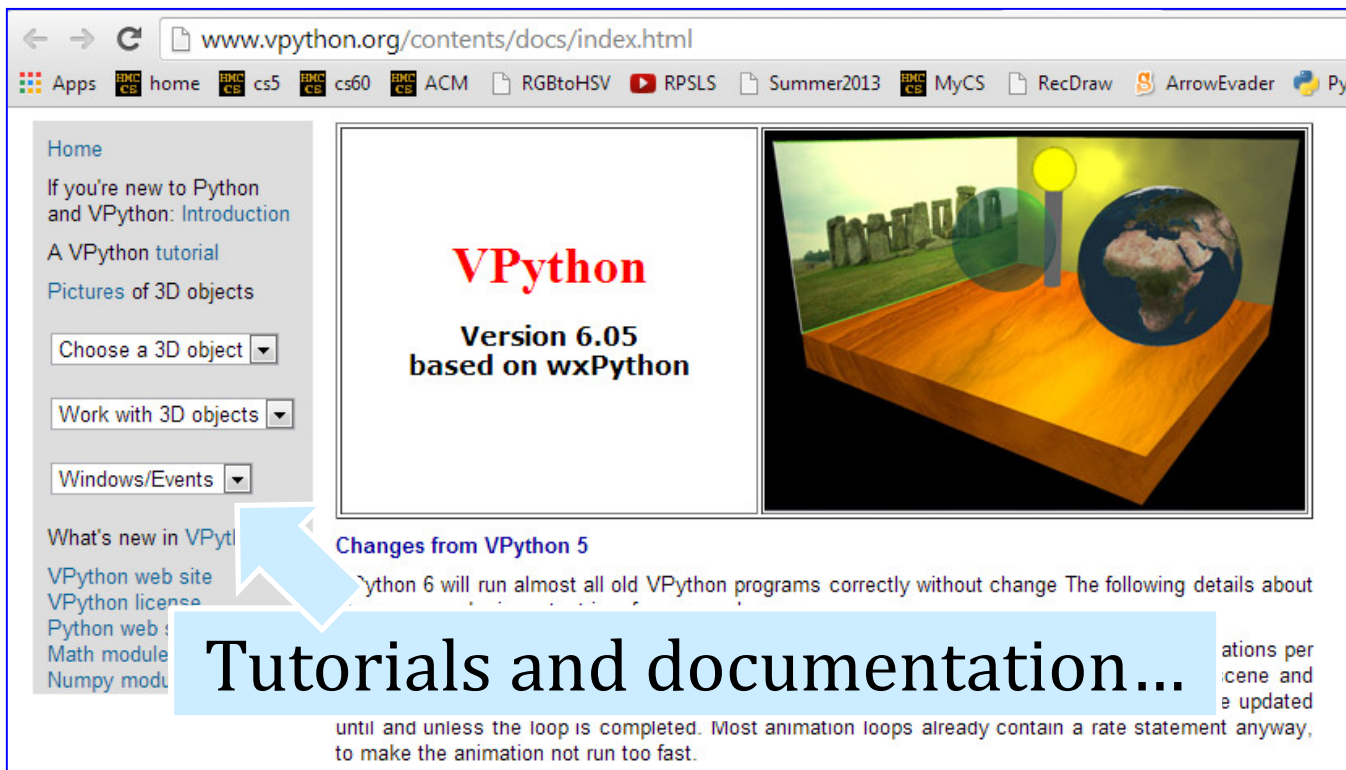
Easily installable for windows... and *mostly* easy on Macs.

# Installing VPython

Windows: [www.vpython.org/contents/download\\_windows.html](http://www.vpython.org/contents/download_windows.html)

Mac: [http://www.vpython.org/contents/download\\_mac.html](http://www.vpython.org/contents/download_mac.html)

*I've tried both of these and they worked so far...*



The screenshot shows the VPython website homepage. The browser address bar displays [www.vpython.org/contents/docs/index.html](http://www.vpython.org/contents/docs/index.html). The page features a sidebar on the left with links to 'Home', 'Introduction', 'tutorial', 'Pictures of 3D objects', and 'What's new in VPython'. The main content area includes the text 'VPython Version 6.05 based on wxPython' and a 3D scene with a yellow sun, a blue Earth, and a green landscape. A blue arrow points from the 'What's new in VPython' link to a large blue box containing the text 'Tutorials and documentation...'. Below this box, there is a paragraph of text: 'until and unless the loop is completed. Most animation loops already contain a rate statement anyway, to make the animation not run too fast.'

**Tutorials and documentation...**

until and unless the loop is completed. Most animation loops already contain a rate statement anyway, to make the animation not run too fast.

# API

... stands for *Application Programming Interface*

a description of how to use a software library

A demo of VPython's API:

```
from visual import *  
c = cylinder()
```

What's **cylinder**?

What's **visual**?

What's **c**?

at least it's not  
*Visual C...*



VPython example API calls: must be from a *file*

# VPython

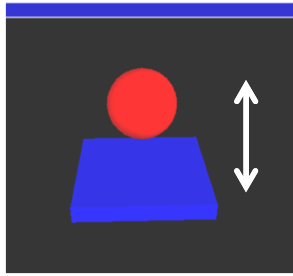
```
from visual import *  
  
c = cylinder()  
  
print "c.pos is", c.pos  
print "c.color is", c.color  
# set the color to color.blue or a tuple  
# set the pos... hard to tell what's happening...  
  
scene.autoscale = False  
  
b = box(pos=(4, 0, 0))  
a = sphere(pos=(0, 0, 4))  
  
while True:  
    rate(100) # limits the loop rate in hz  
    dt = 0.01 # the loop time  
    a.pos += dt*vector(-5, 0, 0)
```

Set up the world  
with 3d objects

more example API calls

Then, run a  
simulation using  
those objects...





# VPython!

Look over this VPython program to determine

- (0) How many tuples appear in this code? \_\_\_\_\_
- (1) How many classes are used here? \_\_\_\_\_
- (2) How many objects are used here? \_\_\_\_\_
- (3) How do *collisions* work?
- (4) How does *physics/gravity* work?

with the Higgs boson!



```
from visual import *
```

```
floor = box( length=4,width=4,height=0.5,color=color.blue )  
ball = sphere( pos=(0,8,0),radius=1,color=color.red )
```

```
vel = vector(0,-1,0)  
dt = 0.01
```

```
while True:  
    rate(100)  
    ball.pos += vel*dt  
  
    if ball.pos.y < ball.radius:  
        vel.y *= -1.0  
    else:  
        vel.y += -9.8*dt
```

What physics is this  
*if/else* doing?

# vectors

act like *vectors*!

```
vel = vector(0, -1, 0)
```

↑  
vel.x

↑  
vel.y

vel.z

← named  
components

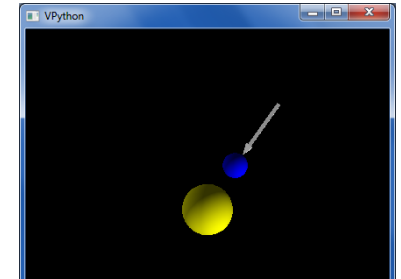
```
vel += 0.01*vector(0, -9.8, 0)
```

↑  
multiplication by a scalar – finally!

```
pos = pos + 0.01*vel
```

↑  
component-by-component addition

# Orbiting



"force arrow" in  
example code...

```
from visual import *
```

```
e = sphere(pos=(0,0,10),color=color.blue) #earth  
s = sphere(color=color.yellow,radius=2)    #sun
```

```
e.vel = vector(5,0,0)          # initial velocity
```

```
RATE = 50
```

```
dt = 1.0/RATE
```

```
k = 70.0                      # G!
```

with *vectors*!

```
while True:
```

```
    rate(RATE)
```

```
    diff = s.pos - e.pos    # vector difference
```

```
    force = k*diff/(mag(diff)**2) # mag
```

```
    e.vel += dt*force        # acceleration d.e.
```

```
    e.pos += dt*e.vel        # velocity d.e.
```

# frames

```
from visual import *
```

```
class Alien:
```

```
    """ This class represents a three-eyed alien object...
    """
```

```
    # The constructor, named __init__ (as always in Python)
```

```
    def __init__(self, init_framepos):
```

```
        """ The constructor creates a frame (container)
            at initial location init_framepos
        """
```

```
        # a frame is VPython's collection of shapes
        # within a single coordinate system
        self.f = frame(pos=init_framepos)
```

```
        # all of these parts are within the frame self.f
```

```
        self.body = sphere(pos=vector(0,0,0),
                             radius=1,
                             color=color.green,
                             frame=self.f)
```

```
        self.left_eye = sphere(pos=self.body.pos + vector(.35, .5, .6),
                                radius=0.20,
                                color=color.white,
                                frame=self.f)
```

```
        self.right_eye = sphere(pos=self.body.pos + vector(-.35, .5, .6),
                                radius=0.20,
                                color=color.white,
                                frame=self.f)
```

# Keyboard events...

```
if scene.kb.keys:                # is there a keyevent?
    s = scene.kb.getkey()        # get keypress
    if s == "p":
        print alien

# things the alien(s) can do!
if s == 'J':    # JUMP!
    alien.f.pos = vector(0, HEIGHT, 0)
    alien.vel = vector(0, 0, 0)        # stop the alien!
    run_gravity = not run_gravity    # fun!
    print "run_gravity is", run_gravity

# move the alien around
if s == "i":
    alien.f.pos += vector(0, 0, 1)
if s == "k":
    alien.f.pos += vector(0, 0, -1)
if s == "j":
    alien.f.pos += vector(-1, 0, 0)
if s == "l":
    alien.f.pos += vector(1, 0, 0)
```

Note that the frame is being moved here ~ this moves ***all*** of the parts!

# Phunky Physics!

```
while True:
    rate(RATE)
    # Here begins PHYSICS!
    if run_gravity == True:
        alien.update(dt)
        alien.check_beach( beach )
    # Here ends physics...
```

in `main()`

```
def update(self, dt):
    """ this defines the physics...
    """
    gravity = -9.8*10
    self.vel += dt*vector(0,gravity,0)
    self.f.pos += dt*self.vel
```

in the `Alien` class

```
def check_beach(self, beach):
    """ checks for a bounce!
    """
    bottom_of_self = self.f.pos.y - self.body.radius
    if bottom_of_self < beach.pos.y:
        self.f.pos.y = beach.pos.y + self.body.radius
        self.vel.y *= -1.0
```

Looking back (before looking forward...)

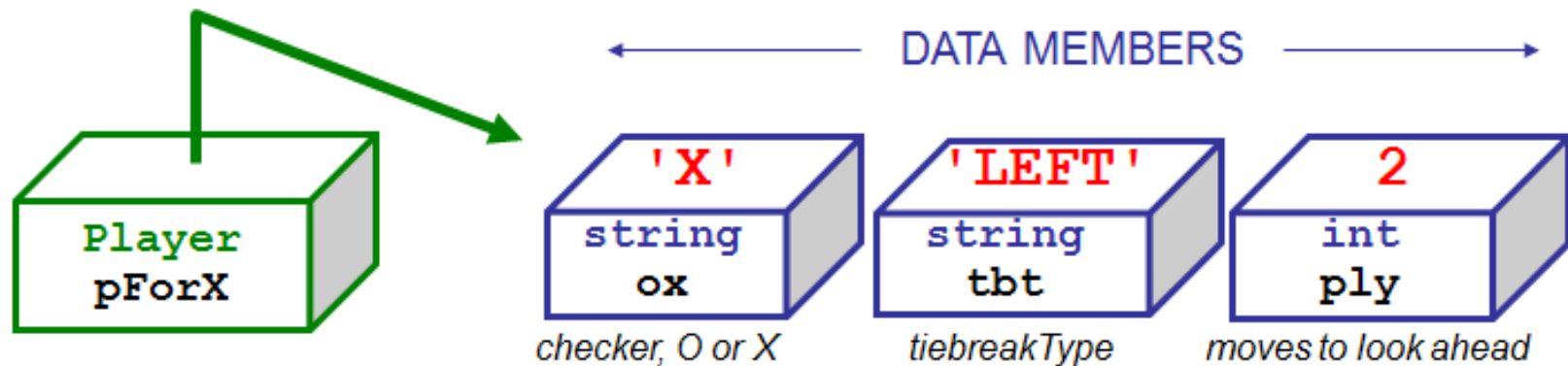
# Lab goals

- (0) Try out VPython!
- (1) Implement air resistance...
- (2) Add at least 1 more dimension
- (3) Add a target and initial velocity
- (4) ***Improve your character!***
- (5) Add scoring or enemies or a moving target, hoops, traps, holes, etc. ~ ***your own game...***



# Next time...

What data does a computer AI player need?



## An AI for Connect Four

## Phunky Physics!

- falls through
- loses energy
- perfect collisions – still imperfect – why?
- air resistance