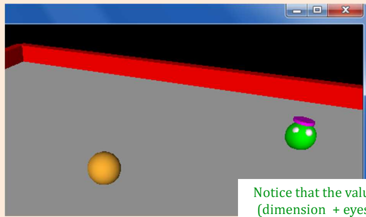
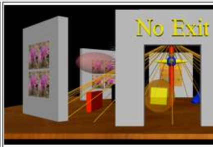
Any memorable prose composed for (well, by) hw#11 ?

This week's `classes`

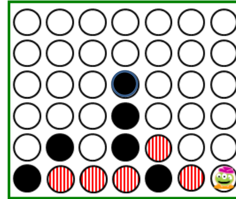


Notice that the value of (dimension + eyes) is conserved!

Three-eyed? This week, we're *3d*'ed!



Reading: 6.7-6.9



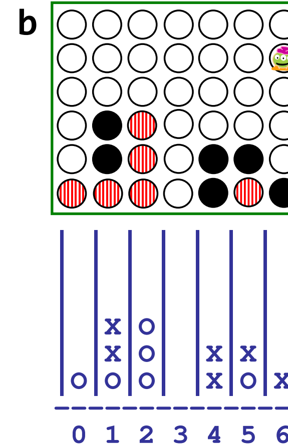
What data does a computer AI player need?

Connect 4
aiMove

Whether it's black's move or red's, they're eye-ing the same column!

Connect 4, Part 2

hw11pr2.py



what methods will help?

`colsToWin(self, ox)`

```
b.colsToWin('O')  # red

b.colsToWin('X')  # black
```

what methods will help?

`aiMove(self, ox)`

```
b.aiMove('O')  # red

b.aiMove('X')  # black
```

`hostGame(self)`

Python features, *motivated* by 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

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

Example of ***default arguments*** 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...***

Python details in VPython...

Functions can have ***default arguments*** and can take ***named arguments***

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

example of an ordinary function call—totally OK `f(4, 2)`

example of *default arguments* `f()`

example using only one *default argument* `f(1)`

example of a *named argument* `f(y = 1)`

API

... stands for *Application Programming Interface*

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

A demo of GlowScript's API:

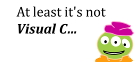
```
GlowScript 3.1 VPython
def main():
    c = cylinder()
    while True:
        rate(10)

# This calls main when the file is run
if __name__ == "__main__":
    main()
```

VPython example API calls

What's `cylinder`?

What's `c`?

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



What's `main`?

What are those last two lines?

VPython

Set up world with 3d objects...

```
b = box()
b.color = color.red
# 1: change colors...
```

color.red or vec(tuple)

```
# 3: add more objects
scene.autoscale = False
c = cylinder(pos = vec(4,0,0))
a = sphere(pos = vec(0,0,4))
```

More objects and API calls

(most will be added ***before*** the while loop)

Then, run a simulation using those objects...

```
while True:
    rate(30)  # times/sec
    print("b.pos is", b.pos)
    # 2: change position
```

```
# 4:  define velocity  (above loop)
# 5:  then, change pos    (in loop)
```

```
if __name__ == "__main__":
    main()
```

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:

http://www.glowscript.org/docs/VPythonDocs/vector.html

vectors

Lots of support... (don't write your own)

The vector Object

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

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

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

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

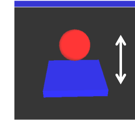
comp(A,B) = A.comp(B) = dot(A,norm(B)), the scalar projection of A along B

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

rotate(A,theta,B) = A.rotate(theta,B) = rotate(vector=A, angle=theta, axis=B), result of rotating A through theta around B

http://www.glowscript.org/docs/VPythonDocs/vector.html

vPython!



Look over this VPython program to determine

(1) How many vPython classes are used in this code? ____
(2) How many vPython objects are used here? ________
(3) What line of code handles *collisions* ?
(4) How does *physics* work? Where is it?
(5) Tricky! How many vectors are here?

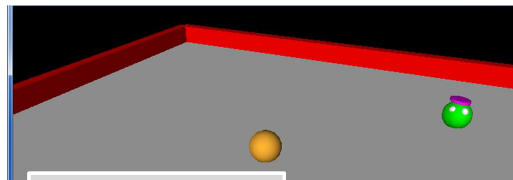
```
def main():
    """docstring!"""
    floor = box(length = 4, width = 4, height = 0.5,
                color = vector(0, 0, 1))
    ball = sphere(pos = vector(0, 8, 0), radius = 1,
                  color = color.red)
    ball.vel = vector(0, -1, 0)
    RATE = 30
    dt = 1.0/RATE

    while True:
        rate(RATE)
        ball.pos += ball.vel*dt

        if ball.pos.y < ball.radius:
            ball.vel.y *= -1.0
        else:
            ball.vel.y += -9.8*dt
```

Let's run it!

what is this += doing?

what is this if/else doing?

Collisions...



point-to-line collisions

Critical!

```
# 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 it 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:

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 = event.key
    ri = randint(0, 10)
    print("key:", key, ri)  # Prints the key pressed

    amt = 0.42   # "Strength" of the keypress's velocity changes
    if key == 'left':
        ball.vel = ball.vel + vector(0, 0, -amt)
    elif key == 'down':
        ball.vel = ball.vel + vector(-amt, 0, 0)
    elif key == 'right':
        ball.vel = ball.vel + vector(0, 0, amt)
    elif key == 'up':
        ball.vel = ball.vel + vector(amt ,0, 0)
    elif key in " rR":
        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 to reset it!

key presses…

```
keys = keysdown()
if keys:
    k = keys[0]
    #print("k is", k)
    if k in '=+': gravity *= 1.1
    if k in '-_': gravity *= 0.9
    if k in '-_+=': print("gravity is", gravity)

    if k in 'gG': ball.color = vec(0, 1, 0) # (r, g, b) from 0 t
    if k in 'rR': ball.color = vec(1, 0, 1)
    if k in 'nN': ball.color = random_color()

    if k == 'down': print('down key')
    if k == 'left': print('left key')
    if k == 'right': print('right key')
    if k == 'up':
        print('up')
        ball.vel.z = 3.0

    if k == 'R':    # Reset!
        ball.vel = vec(0, -1, 0)
        ball.pos = vec(0, 2, 0)
        ball.color = color.red
        gravity = 9.8
```

Key presses…

Named arguments

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

Your name(s) as an argument____________________

`f(3, 1)` →

`f()` →

`f(3)` →

`f(y = 2, x = 1)` →

What will these function calls to `f` return?

None of the above are 42!

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 = -6; x = 60; f(y = x, x = y)** →