This week's classes

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

VPython
3D Programming for Ordinary Mortals

Homework #11, due 4/22

Connect 4 aiMove

Notice that the value of (dimension + eyes) is conserved ~ at 5!

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')

b.colsToWin('X')

what methods will help?

aiMove( self, ox )

b.aiMove('O')

b.aiMove('X')

hostGame( self )
hostGame( self )

b.aiMove( 'X' )

b.colsToWin('O')

b.colsToWin('X')

What methods will help?

More C4 AI ~

and overall AI ~

this Thursday...

Connect 4, Part 2

hw11pr2.py
Final projects

Final CS hw

open-ended
comprehensive
same projects across sections
several choices...

Working in teams of 1-3 is OK

Teams need to work together - *in the same place* - and need to share the work equally...

Pairs/trios are welcome (larger should split)

*Teaming is extra-encouraged* on the final project!
4/22
• "Start" ~ part of hw11

Mon. 4/29
• "Milestone" ~ part of hw12
• project-specific tasks to help w/progress...

 Fri. 5/3
• Final project & short reflection on how to run it and how it went.
• due at 8pm
• Euros ok; grutoring tapers.

When?!

Nice milestone!

It's a kilometer stone, actually!
Final projects

Overview: Project options

- vPool: 3D graphics
- Connect Four: Game AI
- TextID: Text-style matching
- Picobot Project: Genetic algorithms

Start, milestone, and final-project submission

Dates:

- The final version is due Friday, May 3rd at 8pm
- The milestone version is due Monday, April 29th by 11:59pm.
- The starter version is due Monday, April 22nd by 11:59pm.
The projects...

VPool

Picobot!

Vron Hagrid

TextID

C4 AI

4 in a Row
I like poptarts and 42 and spam. Spamful poptarts are like poptartful spams -- and are liked by all! Will _Thanksgiving_ bring spam poptarts?

TextModel

... will have at least five Python dictionaries, e.g.,

```python
File

I like poptarts and 42 and spam. Spamful poptarts are like poptartful spams -- and are liked by all! Will _Thanksgiving_ bring spam poptarts?

{'and': 3, 'poptartful': 1, 'liked': 1, 'spamful': 1, 'like': 2, ':': 1, 'spam': 2, 'i': 1, '42': 1, 'all': 1, 'thanksgiving': 1, 'will': 1, 'bring': 1, 'poptarts': 3, 'spams': 1, 'by': 1, 'are': 2}

{0: 1, 1: 1, 2: 2, 3: 6, 4: 5, 5: 3, 7: 1, 8: 3, 10: 1, 12: 1}

{'and': 3, ':': 1, 'all': 1, 'like': 3, 'thanksgiving': 1, 'spam': 4, 'i': 1, '42': 1, 'by': 1, 'will': 1, 'bring': 1, 'ar': 2, 'poptart': 4}

{12: 1, 5: 1, 7: 1}

punct

{'!': 1, '-': 2, '?': 1, '_': 2, '.': 1}

What are these four other dictionaries counting?!
Naïve Bayes classification

Bayesian spam filtering

From Wikipedia, the free encyclopedia

Bayesian spam filtering (/ˈbɛrizən/ BAY-zee-en; after Rev. Thomas Bayes) is a statistical technique of e-mail filtering. In its basic form, it makes use of a naïve Bayes classifier on bag of words features to identify spam e-mail, an approach commonly used in text classification.

Constructing a classifier from the probability model

The discussion so far has derived the independent feature model, the conditional independence model. The naïve Bayes classifier combines this model with a decision rule to pick the hypothesis that is most probable; this is known as the maximum a posteriori decision rule. The corresponding classifier, a Bayes classifier, is the function defined as follows:

\[
\text{classify}(f_1, \ldots, f_n) = \arg \max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).
\]
Model matching

Suppose we have two trained models:

WS: { "love": 50, "spell": 8, "thou": 42 }

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

JKR: { "love": 25, "spell": 275, "potter": 700 }

how do we handle different-sized texts?

These must have been some really avant-garde texts!
Model *matching*

Suppose we have two trained models:

WS: \{ "love": 50,  
     "spell": 8,  
     "thou": 42 \}  

JKR: \{ "love": 25,  
         "spell": 275,  
         "potter": 700 \}  

Normalize for size

WS: \{ "love": 0.50,  
      "spell": 0.08,  
      "thou": 0.42 \}  

JKR: \{ "love": 0.025,  
       "spell": 0.275,  
       "potter": 0.700 \}  

*Unknown text*: \{ "love": 3,  
                   "thou": 1,  
                   "potter": 2,  
                   "spam": 4 \}
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50,  
      "spell": 0.08,  
      "thou": 0.42 }  

JKR: { "love": 0.025,  
       "spell": 0.275,  
       "potter": 0.700 }

how do we compare the models with an unknown text?  

What's the likelihood of the new model arising from each?

Unknown text: { "love": 3,  
                 "thou": 1,  
                 "potter": 2,  
                 "spam": 4 }

Pretend the words are all independent

There's probably a way to do this!
Suppose we have two normalized models:

WS: \{ "love": 0.50, 
      "spell": 0.08, 
      "thou": 0.42 \} 

JKR: \{ "love": 0.025, 
      "spell": 0.275, 
      "potter": 0.700 \} 

the **WS**-based probability of each word in *Unknown text* 

*Unknown text*: \{ "love": 3, 
                 "thou": 1, 
                 "potter": 2, 
                 "spam": 4 \}

I've got near-zero ideas on this one!
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the **WS**-based probability of each word in *Unknown text*

```
.50 • .50 • .50 • .42 • 0 • 0 • 0 • 0 • 0 • 0 • 0
```

*Unknown text*: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

**correct, but not helpful!**

I've got near-zero ideas on this one!
Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text

\[
\begin{align*}
0.50 \cdot 0.50 \cdot 0.50 \cdot 0.42 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 0.0525
\end{align*}
\]

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

Why is this especially incorrect?

I've got near-zero ideas on this one!
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, 
    "spell": 0.08, 
    "thou": 0.42 }

JKR: { "love": 0.025, 
    "spell": 0.275, 
    "potter": 0.700 }

the WS-based probability of each word in Unknown text

for missing words, use half the smallest value – across both normalized models!

Unknown text: { "love": 3, 
    "thou": 1, 
    "potter": 2, 
    "spam": 4 }

\[ 0.50 \times 0.50 \times 0.50 \times 0.42 = 1.57 \times 10^{-13} \]
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text

for missing words, use half the smallest value – across both normalized models!

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

\[ \log(0.50 \cdot 0.50 \cdot 0.50 \cdot 0.012 \cdot 0.012 \cdot 0.012 \cdot 0.012 \cdot 0.012 \cdot 0.012) = 1.57 \times 10^{-13} \]

too small!

take logs of everything!

half of $\epsilon$!
Model matching

Suppose we have two normalized models:

WS: `{"love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: `{"love": 0.025, "spell": 0.275, "potter": 0.700 }

Unknown text: `{"love": 3, "thou": 1, "potter": 2, "spam": 4 }

for missing words, use half the smallest value – across both normalized models!

3*log(.50) + log(.42) + 2*log(.012) + 4*log(.012) = -29.48

take logs of everything!

OK!

half of $\epsilon$!
Model matching

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text

3*\log(0.025) + \log(0.012) + 2*\log(0.7) + 4*\log(0.012) = -33.89

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

this looks close...
Suppose we have two normalized models:

WS: { "love": 0.50,  
      "spell": 0.08,  
      "thou": 0.42 }  

JKR: { "love": 0.025,  
       "spell": 0.275,  
       "potter": 0.700 }  

Unknown text:
{ "love": 3,  "potter": 2,  
  "thou": 1,  "spam": 4 }  

-29.48  

-33.89  

the (much) better match...
Naïve Bayes classification

Bayesian spam filtering

From Wikipedia, the free encyclopedia

Bayesian spam filtering (ˈberzən/ BAY-zən; after Rev. Thomas Bayes) is a statistical technique of e-mail filtering. In its basic form, it makes use of a naïve Bayes classifier on bag of words features to identify spam e-mail, an approach commonly used in text classification.

Constructing a classifier from the probability model

The discussion so far has derived the independent feature model, that is, the naive Bayes model. The naive Bayes classifier combines this model with a decision rule. One is to choose the hypothesis that is most probable; this is known as the maximum a posteriori decision rule. The corresponding classifier, a Bayes classifier, is the function classify.

$$classify(f_1, \ldots, f_n) = \arg \max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).$$

Don't take these formulas too seriously...
Naïve Bayes classification

Bayesian spam filtering

From Wikipedia, the free encyclopedia

Bayesian spam filtering (ˈbeɪzɪən/ BAY-zee-ən; after Rev. Thomas Bayes) is a form of probabilistic classifier. It is based on Bayes' theorem and assumes that the presence (or absence) of a particular feature of an instance is unrelated to the presence (or absence) of any other feature, given the class label of the instance. The classifier works by computing the posterior probability of each class label given the features of the instance and then selecting the label with the highest posterior probability.

\[
Pr(S|W) = \frac{Pr(W|S) \cdot Pr(S)}{Pr(W|S) \cdot Pr(S) + Pr(W|H) \cdot Pr(H)}
\]

This quantity is called "spamicity" (or "spaminess") of the word "replica", and can be computed. The number of occurrences of "replica" in an email can be used to classify it as spam.

The discussion so far has derived the independent feature model, that is, the naive Bayes classifier. The naive Bayes classifier combines this model with a decision rule. One way to make a decision is to choose the hypothesis that is most probable; this is known as the maximum a posteriori decision rule. The corresponding classifier, a Bayes classifier, is the function classifying $f_1, \ldots, f_n$ as:

\[
\text{classify}(f_1, \ldots, f_n) = \arg\max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i|C = c).
\]

Don't take these formulas too seriously… done!
The projects...

VPool

Harry Potter

Voldemort

TextID

C4 AI

Picobot!
The Picobot project

(1) Implement Picobot in Python
(2) *Train Python to write successful Picobot programs!*

talk about going *full circle*...
Picobot, *behind the curtain*...

What data structures (classes) might be helpful in implementing Picobot?
Picobot's classes

```python
class Program:

    # What in Python could most usefully hold all of these rules?

    # What type should self.rules be?

    # Dictionary!
```

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 xxxx</td>
<td>N 0</td>
</tr>
<tr>
<td>0 Nxxx</td>
<td>W 0</td>
</tr>
<tr>
<td>0 NxWx</td>
<td>S 0</td>
</tr>
<tr>
<td>0 xxWx</td>
<td>S 0</td>
</tr>
<tr>
<td>0 xxWS</td>
<td>E 0</td>
</tr>
<tr>
<td>0 xxxxS</td>
<td>E 0</td>
</tr>
<tr>
<td>0 xExS</td>
<td>N 0</td>
</tr>
<tr>
<td>0 xExx</td>
<td>N 0</td>
</tr>
<tr>
<td>0 NExx</td>
<td>S 1</td>
</tr>
<tr>
<td>1 xxxx</td>
<td>S 1</td>
</tr>
<tr>
<td>1 Nxxx</td>
<td>E 1</td>
</tr>
<tr>
<td>1 NxWx</td>
<td>E 1</td>
</tr>
<tr>
<td>1 xxWx</td>
<td>N 1</td>
</tr>
<tr>
<td>1 xxWS</td>
<td>N 1</td>
</tr>
<tr>
<td>1 xxxxS</td>
<td>W 1</td>
</tr>
<tr>
<td>1 xExS</td>
<td>W 1</td>
</tr>
<tr>
<td>1 xExx</td>
<td>S 1</td>
</tr>
<tr>
<td>1 NExx</td>
<td>W 0</td>
</tr>
</tbody>
</table>
Picobot's classes

**class** Program:

What in Python could most usefully hold all of these *rules*?

What type should `self.rules` be?

```python
self.rules[ (1,"NExx") ] = ("W",0)
```
Picobot's classes

What type in Python could most usefully hold the *environment*?

class World:

What *class* that you've already written will be most similar to Picobot's *World*?

What will *self.room* be?
Picobot's classes

What type in Python could most usefully hold the *environment*?

```python
class World:
    # class implementation

What **class** that you've already written will be most similar to Picobot's *World*?

What will **self.room** be?

The same as the Connect-Four board's **self.data**!

**Wall:** ⊞
**Visited:** ○
**Picobot:** ☻
```

*a list-of-lists-of-one-character-strings....*
Picobot's project

First, build an **ASCII simulation**

Current State: 1
Current Rule: 1 N*W* -> X 2

```
+ + + + + + + + +
+ o + + o + + + +
+ o o o o o o + +
+ + + + + + + + +
+ + + + + + + + +
+ o o o o o o + +
+ + + + + + + + +
+ o o o o o o + +
+ + + + + + + + +
+ + + + + + + + +
+ P + + + + + + +
+ + + + + + + + +
```

Picobot started here...

and is now here...

Your actual ASCII is likely to be more monochromatic!

---

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrIPf8
Picobot's project

**Current State:** 1
**Current Rule:** $1 \text{ N*W*} \rightarrow \text{ X 2}$

---

First, build an *ASCII simulation*

---

- Picobot started here...
- and is now here...

---

Your actual ASCII is likely to be more monochromatic!

---

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrPlf8
Picobot's project

Current State: 1
Current Rule: 1 N*W* -> X 2

First, build an ASCII simulation
then, evolve it...

with your own genetic algorithm

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrlPif8

Your actual ASCII is likely to be more monochromatic!
Program *evolution*

An example of *genetic algorithms*, which are used for optimizing *hard-to-describe functions* with *easily-splittable solutions*.

Start with a population of, say, ~200 *random* Picobot programs...
Program *evolution*

An example of *genetic algorithms*, which are used for optimizing hard-to-describe functions with easily-splittable solutions.

Then, *measure* the fitness of all of those programs.

Measure? *How??*
program p1
fitness = 0.03

program p2
fitness = 0.05

mate + mutate the fittest rulesets
to create a new generation of ~200 programs...

program c1
fitness = 0.19

What the goal?
Repeat this "survival of the fittest" process for many generations...

... and by the end, your Python code should/will have evolved a much more capable Picobot program!

fitness = 0.90
VPool

Today's lab!

Not 3... but 4!

Picobot!
Let's play!

3d graphics-based game using VPython

I'll take your cue.

Physics engine...

... it's not really very constrained at all!

A few constraints...

need ≥4 physically interacting objects

allow the user to direct 1+ objects, either by keyboard or mouse or both

needs a game goal + be winnable!

must detect some "linear" and some "spherical" collisions and implement their results on the motion of the objects
Example
vProjects...

Do you see the pool table!? the pool balls? the alien?
Python features, *motivated* by VPython...

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

\[
T = (4, 2) \quad x = (1, 0, 0)
\]

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

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

eamples of *default and named inputs* in a function definition
Lists that use parentheses are called **tuples**:

\[
T = (4, 2)
\]

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*

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

Example of default input values for `x` and `y`

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

Example of default input values for `x` and `y`

Function inputs *look like* tuples, but they're not quite the same…
Default – *and named – inputs!*

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

```python
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) = ________________

---

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

- `f(3, 1)` returns `3 + 3 = 6`
- `f(3)` returns `3 + 3*3 = 12`
- `f(y=4, x=2)` returns `2 + 12 = 14`
- `f()` returns `2 + 3*3 = 11`

None of the above are 42!

---

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

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

---

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

These are tuples – they work like lists!

- `f((1,0))` returns `1 + 3*0 = 1`
- `f((1,0,1,0,1,0))` returns `1 + 3*0 = 1`
- `f((1,0,1,0,1,0))` returns `1 + 3*0 = 1`
- `f((1,0,1,0,1,0))` returns `1 + 3*0 = 1`

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

---

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

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

---

but they all share a factor with it! - Eli B. '17
```
def f(x=2, y=11):
    return x + 3*y
```

What will these function calls to `f` return?

- `f(3,1)`
- `f(3)`
- `f(y=4, x=2)`
- `f()`

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

What will these function calls to `f` return?

- `f(3, 1)` returns `6`
- `f(3)` returns `36`
- `f(y=4, x=2)` returns `None` (None of the above are 42!)
- `f()` returns `35`

What call to `f` returns the string 'Lalalalalala'?

- `f('Lala', 'la')`

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

- `f(( ), (1,0))` returns `(1,0,1,0,1,0)` (it's only four characters, too!)

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

- `-14`

These are tuples – they work like lists!
```python
def f(x=2, y=11):
    return x + 3*y
```

- `f(3, 1)` returns `6`
- `f(3)` returns `6`
- `f(y=4, x=2)` should return `34`

What will these function calls to `f` return?

None of the above are 42!

Try this on the back page first…

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

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

What is `f((3,1), (1,0))`?

These are tuples – they work like lists!

It's only four characters, too!

**Extra…** what does this return?  

```python
y = 60; x = -6; f(y=x, x=y)
```
Where were we...?

VPython ~ GlowScript!

built by and for physicists to simplify 3d simulations

lots of available classes, objects and methods in its API

www.glowscript.org/
Using GlowScript / vPython...

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.

Documentation

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

- 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

examples...
documentation...
API

... stands for Application Programming Interface

A demo of vPython's API:

```python
# 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
```

vPython example API call(s)

What's `box`?
What's `color`?
What's `vector`?
Getting used to everything!
API... stands for Application Programming Interface

constructors + methods!

shapes + docs!

cool stuff...
API stands for Application Programming Interface

constructor + default arguments; data!

Here is how to create a box object:

```python
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:

```python
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 coordinate system.
vectors

\[ \mathbf{b}.\text{vel} = \text{vector}(1,0,0) \]

\[ \mathbf{b}.\text{pos} = \text{vector}(0,0,0) \]

\[ \mathbf{b}.\text{pos} = \mathbf{b}.\text{pos} + \mathbf{b}.\text{vel} \times 0.2 \]

\textit{let's compare with tuples...}
The `vector` Object

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

```python
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,

```python
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, 20, 30).
- 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
# 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)
Vector functions

The following functions are available for working with vectors:

\[
\text{mag}(A) = A . \text{mag} = |A|, \text{ the magnitude of a vector}
\]

\[
\text{mag2}(A) = A . \text{mag2} = |A|^{2}, \text{ the vector's magnitude squared}
\]

\[
\text{norm}(A) = A . \text{norm}() = A / |A|, \text{ a unit vector in the direction of the vector}
\]

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

\[
\text{dot}(A,B) = A . \text{dot}(B) = A \cdot B, \text{ the scalar dot product between two vectors}
\]

\[
\text{cross}(A,B) = A . \text{cross}(B), \text{ the vector cross product between two vectors}
\]

\[
\text{diff_angle}(A,B) = A . \text{diff_angle}(B), \text{ the angle between two vectors, in radians}
\]

\[
\text{proj}(A,B) = A . \text{proj}(B) = \text{dot}(A, \text{norm}(B)) \times \text{norm}(B), \text{ the vector projection of } A \text{ along } B
\]

\[
\text{comp}(A,B) = A . \text{comp}(B) = \text{dot}(A, \text{norm}(B)), \text{ the scalar projection of } A \text{ along } B
\]

\[
A . \text{equals}(B) \text{ is True if } A \text{ and } B \text{ have the same components (which means that they have the same magnitude and the same direction)}
\]

\[
\text{vec.random}() \text{ produces a vector each of whose components are random numbers in the range -1 to +1}
\]
Rotating a vector

There is a function for rotating a vector:

\[ v2 = \text{rotate}(v1, \, \text{angle}=a, \, \text{axis}=\text{vec}(x,y,z)) \]

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 \( v2 = v1.\text{rotate}(\text{angle}=\theta, \text{axis}=\text{vec}(1,1,1)) \). There is also a rotate capability for objects.

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

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

- \( \text{radians}(360) \) is equivalent to \(2\pi\)
- \( \text{degrees}(2\pi) \) is equivalent to 360
JavaScript is a web development language that can be used alongside HTML and CSS to make dynamic web pages. It allows for content to be updated, for animations and other multimedia content to be displayed on a webpage.
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?

Line ~ wall at x=10
Spherical collisions

0 **Zeroth** approximation:
   Stop \( q \). *Undo any overlap.*
   Make \( r.\text{vel} = q.\text{vel} \).

1 **First** approximation:
   Stop \( q \). *Undo any overlap.*
   Compute \( d = r.\text{pos} - q.\text{pos} \)
   Make \( r.\text{vel} = d \)

2 **Second** approximation:
   Same as *first*, but
   Make \( q.\text{vel} = d^\perp \), at 90° from \( d \)

Reality is just one eye away!
Lab 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. ~ *final project*…

one of the final project options
# 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)
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:

```plaintext
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.
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:

```python
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)
GlowScript / vPython examples...
GlowScript / vPython examples...

Try out vPython in lab this week!

~ if you enjoy it, consider it for a final project