4/1/15!
4/1/12!
New! Introducing Gmail Paper

Everyone loves Gmail. But not everyone loves *email*, or the digital era. What ever happened to stamps, filing cabinets, and the mailman? Well, you asked for it, and it’s here. We’re bringing it back.

A New Button
Now in Gmail, you can request a physical copy of any message with the click of a button, and we’ll send it to you in the mail.

Simplicity Squared
Google will print all messages instantly and prepare them for delivery. Allow 2-4 business days for a parcel to arrive via post.

Total Control
A stack of Gmail Paper arrives in a box at your doorstep, and it’s yours to keep forever. You can read it, sort it, search it, touch it. Or even move it to the trash—the real trash. (Recycling is encouraged.)

Keep it Secret, Keep it Safe
Google takes privacy very seriously. But once your email is physically in your hands, it’s as secure as you want to make it.

Learn more about Gmail Paper
IST 338 Rules!

Rule #1: Don't follow this rule.

Simple rules can yield complex results!

Two problems - two rules each

HW 9 (lab + 1 problem)
due Fri. 4/10

Final projects: a look ahead...

Three-in-a-row? Aliens rule at this...
Dear Professor Dodds,

I know you will like these attachments… no regret for next semester!

Best,
Sebastian
Dear Professor Dodds,

I know you will like these attachments... no regret for next semester!

Best,
Sebastian
In CS, \textit{rules} rule!

(1a) Lists are handled "by reference"

\begin{itemize}
  \item \begin{itemize}
    \item \textit{Reference} "Pointer"
    \end{itemize}
  \end{itemize}

\texttt{L} = [5, 42, 'hi']

(1b) Numbers and strings are handled "by value"

\texttt{s} = 'hi'
Python's two methods for handling data

**Reference** vs. **Value**

**Lists** are handled **by reference**: 
$L$ really holds a *memory address*

**Numeric data** and **strings** are handled **by value**: imagine they *hold* the data

$L = [5, 42, 'hi']$

$s = 'hi'$

$x = 7$
In CS, rules rule!

(1a) Lists are handled "by reference"

\[
L = [5, 42, 'hi']
\]

(1b) Numbers and strings are handled "by value"

(2) Inputs pass to functions "by copy"

The contents of the variable's "box" in memory are copied.

in main:
```
fav = 7
f(fav)
```

def f(x):
```
x = 7
```

x is a copy of fav
Python functions: *pass by copy*

```python
def conform(fav):
    fav = 42
    return fav

def main():
    print "Welcome!"
    fav = 7
    fav = conform(fav)
    print "My favorite # is", fav
```

*Pass by copy*: When a function modifies its parameter, the original parameter remains unchanged. However, if a copy of the parameter is created, the original parameter is left unmodified.
Python functions: *pass by copy*

```python
def main()
    print "Welcome!"
    fav = 7
    fav = conform(fav)
    print "My favorite # is", fav

def conform(fav)
    fav = 42
    return fav
```

"pass by copy" means the contents of `fav` are copied to `fav`
Try it!  

Rules rule!?  

Trace each f'n. What will main1, main2, and main3 print?

```
def conform1(fav)
    fav = 42
    return fav
```

```
def conform2(L)
    L = [42,42]
    return L
```

```
def conform3(L)
    L[0] = 42
    L[1] = 42
```

```
def main1()
    fav = 7
    conform1(fav)
    print fav
```

```
def main2()
    L = [7,11]
    conform2(L)
    print L
```

```
def main3()
    L = [7,11]
    conform3(L)
    print L
```

Notice that there are NO assignment statements after these function calls! The return values aren't being used...
Lists are *Mutable*

You can change the **contents** of lists in functions that take those lists as input.

- Lists are MUTABLE objects

Those changes will be visible **everywhere**.

Numbers, strings, etc. are IMMUTABLE – they can't be changed, only reassigned.
Differing approaches to rules ...
Engineers believe equations approximate reality;
Engineers believe equations approximate reality; **Physicists** believe reality approximates equations...

http://www.youtube.com/watch?feature=player_embedded&v=WbaH52JI3So

Image forensics' verdict: **Fake!**

Not a parabola, so not a real shot!
Engineers believe equations approximate reality; 
Physicists believe reality approximates equations...

Mathematicians don't care either way!

A solid sphere can be split into 5 parts and rigidly reassembled into two spheres the same size as the original

Banach-Tarski paradox

the parts are "mist"
Engineers believe equations approximate reality; Physicists believe reality approximates equations...

Mathematicians don't care either way!

In CS? Don't like reality? **Build a new one!**

why settle for gears, when you could have **fractal** gears?
Engineers believe equations approximate reality; Physicists believe reality approximates equations...

*Mathematics reasons about structural rules...*  
... and CS reasons about *procedural* ones.

**Math worldview**

- Axioms
- Data
- Definitions
- proofs
- Insights, tools, mathematical truths

**CS worldview**

- lists
- for
- while
- variables
- arithmetic operations
- if/else
- programs
- Insights, tools, algorithms
2D data!
Lists ~ 2D data

A = [ 42, 75, 70 ]

1D lists are familiar – but lists can hold ANY kind of data – **including lists!**
Lists ~ 2D data

\[ A = \begin{bmatrix} [1,2,3,4], & [5,6], & [7,8,9,10,11] \end{bmatrix} \]

Where's 3? \( \text{len}(A[0]) \) \( \text{len}(A) \) Replace 10 with 42.
Rectangular 2D data

\[ A = \begin{bmatrix} [1,2,3,4], [5,6,7,8], [9,0,1,2] \end{bmatrix} \]

To try...

- What is \( A[1][2] \)?
- What does each component of \( A[1][2] \) mean?

Using `len`, how many rows does \( A \) have, in general?

Using `len`, how many columns does \( A \) have, in general?
Try it...

```python
def mystery(A):
    """ what happens to A ? """
    NROWS = len(A)
    NCOLS = len(A[0])
    for r in range(0,NROWS):
        for c in range(0,NCOLS):
            if A[r][c] == 4:
                A[r][c] = 1
            else:
                A[r][c] += 1
```

Write in the resulting values in `A`:

```python
A = [[4, 2, 2, 2],
     [2, 2, 4, 4],
     [2, 4, 4, 2]]
```

Starting with the 2d array `A` shown above, write the values in `A` after running the code?

After – with the code as written...
**def inarow_2east(A):**

""" what happens to A ? """

NROWS = len(A)
NCOLS = len(A[0])

**for r in range( 0,NROWS ):**
    **for c in range( 0,NCOLS ):**

How would you *change the code* above to produce a True where the original value is equal *to its neighbor to the right*?! 

(False, if no neighbor or a different neighbor.)
First, try it by eye...  

... then, on hw9pr2, by Python!

\[
A = \begin{bmatrix}
\text{row 0} & [ & ' ', 'X', 'O', ' ', 'O' ] , \\
\text{row 1} & [ & 'X', 'X', 'X', 'O', 'O' ] , \\
\text{row 2} & [ & ' ', 'X', 'O', 'X', 'O' ] , \\
\text{row 3} & [ & 'X', 'O', 'O', ' ', 'X' ]
\end{bmatrix}
\]

\text{inarow\_3east}('X', 1, 0, A)
First, try it by eye... ... then, on hw9pr2, by Python!

\[
A = \begin{bmatrix}
  [' ', 'X', 'O', ' ', 'O'], \\
  ['X', 'X', 'X', 'O', 'O'], \\
  [' ', 'X', 'O', 'X', 'O'], \\
  ['X', 'O', 'O', ' ', 'X']
\end{bmatrix}
\]

\begin{align*}
inarow\_3east('X', 1, 0, A) & \quad \rightarrow \quad True \\
inarow\_3south('O', 0, 4, A) & \\
inarow\_3southeast('X', 2, 3, A) & \\
inarow\_3northeast('X', 3, 1, A) &
\end{align*}

the data doesn't wrap around
The fantastic combinations of John Conway's new solitaire game "life"

by Martin Gardner

*Scientific American* 223 (October 1970): 120-123.
Lab Problem: *Conway's Game of Life*

**Grid World**
- red cells are "alive"
- white cells are empty

**Evolutionary rules**
- Everything depends on a cell's eight neighbors
- Exactly 3 neighbors give birth to a new, live cell.
- Exactly 2 or 3 neighbors keep an existing cell alive.
- Any other # of neighbors and the central cell dies...
Lab Problem: *Conway's Game of Life*

**Grid World**
- red cells are "alive"
- white cells are empty

**Evolutionary rules**
- Everything depends on a cell's eight neighbors
- Exactly 3 neighbors give birth to a new, live cell.
- Exactly 2 or 3 neighbors keep an existing cell alive.
- Any other # of neighbors and the central cell dies...
Lab Problem: *Conway's Game of Life*

**Grid World**
- red cells are "alive"
- white cells are empty

**Evolutionary rules**
- Everything depends on a cell's eight neighbors
- Exactly 3 neighbors give birth to a new, live cell.
- Exactly 2 or 3 neighbors keep an existing cell alive.
- Any other # of neighbors and the central cell dies...
Lab Problem: *Conway's Game of Life*

**Grid World**
- red cells are alive
- white cells are empty

**Evolutionary rules**
- Everything depends on a cell's eight neighbors
- Exactly 3 neighbors give birth to a new, live cell.
- Exactly 2 or 3 neighbors keep an existing cell alive.
- Any other # of neighbors and the central cell dies...

What's next?
Lab Problem: *Creating life*

For each cell…

- 3 live neighbors – **life**!
- 2 live neighbors – **same**
- 0, 1, 4, 5, 6, 7, or 8 live neighbors – **death**

- computed all at once, not cell-by-cell, so the ? at left does NOT come to life!

Lab Problem: *Creating life*

`next_life_generation(A)`

old generation is the input, A returns the next generation
Lab Problem: *Creating life*

Stable configurations:
- "rocks"
- "plants"
- "animals"

Periodic
- "plants"
  - period 2
  - period 3

Self-propagating
- "animals"
- glider
Lab Problem: *Creating life*

Many life configurations expand forever...

What is the largest amount of the life universe that can be filled with cells?

How sophisticated can Life-structures get?

www.ibiblio.org/lifepatterns/
Ex Cr: the Mandelbrot Set

Consider an update rule for all complex numbers $c$

$z_0 = 0$

$z_{n+1} = z_n^2 + c$
Nothing's too complex for Python!

```python
>>> c = 3 + 4j
>>> c.real
3.0
>>> c.imag
4.0
>>> abs(c)
5.0
```
Mandelbrot Definition

Consider an **update rule** for all complex numbers $c$

\[
\begin{align*}
    z_0 &= 0 \\
    z_{n+1} &= z_n^2 + c
\end{align*}
\]

*Small values of $c$ keep the sequence near the origin, $0+0j$.*

*Some "stick around": oscillate or converge*
Consider an **update rule** for all complex numbers $c$

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$

*Small values of $c$ keep the sequence near the origin, $0+0j$.*

*Other values of $c$ make the sequence head to infinity.*
Consider an update rule for all complex numbers $c$

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$
Consider an \textit{update rule} for all complex numbers $c$

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$
Mandelbrot Set ~ points that stick around

The shaded area are points that do not diverge for \( z = z^{**2} + c \)
The black pixels are points that do **not** diverge for \( z = z^{**2} + c \)

- **connected**
- **finite area**
- \( \infty \) **perimeter!**
Complex things always consisted of simple parts...
Before the M. Set, complex things were made of simple parts:

Chaos!

This was a "naturally occurring" object where zooming uncovers more detail, not less:

not self-similar but quasi-self-similar

http://www.youtube.com/watch?v=0jGaio87u3A
The black pixels are points that do not diverge for $z = z^{**2} + c$

What are these colors?
The black pixels are points that do *not* diverge for \( z = z^{**2} + c \).

What are these colors?
Numbers in yellow indicate the number of dendrites or spiral arms found in each region, and in the corresponding Julia fractals for each region.
In the Seahorse Valley....
Happy Mandelbrotting!

www.cs.hmc.edu/~jgrasel/projects

http://www.youtube.com/watch?v=0jGaio87u3A
Tis the season for final projects...

Today is about the our final projects + it's a sales pitch for the three possible options:

- vPool
- TextID
- Picobot

I've got my eyes on some of these projects!

Eye'll bet!
Final projects

Final CS assignment

- open-ended
- comprehensive
- same projects for black/gold
- three choices...

Working solo or in a pair is OK

Pairs need to work together - *in the same place* - and they need to share the work equally...
Project options...

Design-your own (by 4/17...)

See the IST 338 Final Projects page...
Pre-planned possibilities

at least to think about...
The Picobot project

Big ideas:

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

talk about going full circle...
Picobot *returns!*

What data structures 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?

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

```python
class World:

What in Python could most usefully hold the environment?

What type should self.room be?
```

Wall: ✦
Visited: ○
Picobot: P
Picobot's classes

```python
class World:
```

What in Python could most usefully hold the `environment`?

What type should `self.room` be?

2d data just as we've been using this week!
The Picobot project

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

First build an \textit{ASCII simulation}
Program evolution

An example of genetic algorithms, used for optimizing hard-to-describe functions.

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

An example of *genetic algorithms*, used for optimizing hard-to-describe functions.

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

Evaluate? *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
Repeat this "survival of the fittest" process for many generations...

... and by the end, your Python code should have evolved a much more capable Picobot program!
Project options...
Though Robin Ellacott’s twenty-five years of life had seen their moments of drama and incident, she had never before woken up in the certain knowledge that she would remember the coming day for as long as she lived.
How Robert Galbraith was found to be JK Rowling

SARAH PAVIS • AUG 08 2013

I was given e-text copies of Cuckoo to compare against Rowling’s own The Casual Vacancy, Ruth Rendell’s The St. Zita Society, P.D. James’ The Private Patient and Val McDermid’s The Wire in the Blood. [...] 

I actually ran four separate types of analyses focusing on four different linguistic variables. While anything can in theory be an informative variable, my work focuses on variables that are easy to compute and that generate a lot of data from a given passage of language. One variable that I used, for example, is the distribution of word lengths. Each novel has a lot of words, each word has a length, and so one can get a robust vector of % of the words in this document have exactly letters. Using a distance formula (for the mathematically minded, I used the normalized cosine distance formula instead of the more traditional Euclidean distance you remember from high school), I was able to get a measurement of similarity, with 0.0 being identity and progressively higher numbers being greater dissimilarity.
the TextID project

Big ideas:

(1) Build *lexical* models of bodies of text...
(2) *Create a similarity score that defines* Rowlingness vs. Shakepearity

NYTimes-iness vs. WSJournalicity

*Big Bang Theory?* vs. *Arrested Development*

*your own choice of two comparisons...*
TextID **model**

word-frequencies

WS: `{ "love": 50, "spell": 8, ... }

JKR: `{ "love": 11, "spell": 277, ... }

stem-frequencies

WS: `{ "lov": 612, "spell": 9, ... }

JKR: `{ "lov": 98, "spell": 306, ... }

word-length-freq.'s / sentence-length-freq.'s

WS: `{ 4: 7042, 5: 6203, ... }

JKR: `{ 4: 980, 5: 42005, ... }

*all will be Python dictionaries...*
Processing steps: one idea

Split the text into words...
Clean the words ...
Remove all non-alphabetic characters
Find sentence lengths using punctuation
Create dictionaries of the words and their lengths

Stem the words

Create a dictionary of word stems

You're ready to match against another model!
Stemming

An algorithm that outputs the root of the input word.

- `stem( "party" )` → "parti"
- `stem( "parties" )` → "parti"
- `stem( "love" )` → "lov"
- `stem( "loving" )` → "lov"

These don't have to be words, just stems...
Stemming

\[
\text{stem( "party" ) } \rightarrow \text{ "parti"}
\]
\[
\text{stem( "parties" ) } \rightarrow \text{ "parti"}
\]
\[
\text{stem( "love" ) } \rightarrow \text{ "lov"}
\]
\[
\text{stem( "loving" ) } \rightarrow \text{ "lov"}
\]

Use the Porter Stemming Algorithm!

Or \textit{tries} to output the root!

these don't have to be words, just \textbf{stems}...
Model matching

Suppose we have two dictionaries:

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

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

"potter" is not here.
"thou" is not here.

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

How could we give a match score for this New dictionary against each one above?
Naïve Bayes classification

**WS:** \{ "love": 50, "spell": 8, "thou": 42 \}  
**JKR:** \{ "love": 25, "spell": 275, "potter": 700 \}

"potter" is not here.  
"thou" is not here.

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

```
<table>
<thead>
<tr>
<th>Word</th>
<th>WS Score</th>
<th>JKR Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>love</td>
<td>50/100</td>
<td>25/1000</td>
</tr>
<tr>
<td>spell</td>
<td>8/100</td>
<td>275/1000</td>
</tr>
<tr>
<td>thou</td>
<td>42/100</td>
<td>0/1000</td>
</tr>
<tr>
<td>potter</td>
<td>0/100</td>
<td>700/1000</td>
</tr>
</tbody>
</table>
```

**Score vs. WS:** Score = 0  
**Score vs. JKR:** Score = 0

- **Multiply each word's probability as if they were all independent**
Naïve Bayes classification

WS: { "love": 50, "spell": 8, "thou": 42 }
JKR: { "love": 25, "spell": 275, "potter": 700 }

"potter" is not here.
"thou" is not here.

New: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }
Naïve Bayes classification

Bayesian spam filtering

Bayesian spam filtering (\textit{bay-zee-en}; after Rev. Thomas Bayes) is a statistical technique of e-mail filtering. In its basic form, it makes use of a naive 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 probability model. The naive Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posteriori or MAP decision rule. The corresponding classifier, a Bayes classifier, is the function \texttt{classify} defined as follows:

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

Bayesian spam filtering

From Wikipedia, the free encyclopedia

Bayesian spam filtering (ˈbeɪzɪən/ 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 the word occurrence frequencies to identify spam e-mail or, more generally, to filter out unwanted messages from being delivered to the user's inbox. The spam filter is regarded as simple to implement and effective at detecting spam. It is thus widely used in spam detection.

The discussion considers a model in which there is an underlying Bernoulli distribution that determines whether a given word is present or absent in a particular document, and a classifier that takes this Bernoulli distribution as input and predicts the probability that the document is in a given category. The classifier uses Bayes' theorem to compute this probability. In the case of naive Bayes, the independence assumption is that the presence of individual words is independent of all other words.

This quantity is called "spamicity" (or "spaminess") of the word "replica", and can be computed. The number

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

Don't take these formulas too seriously...
Project options...

VPool

TextID

Picobot!
3d graphics-based game using VPython

I'll take your cue...

Let's play!

vPool

not really very constrained at all!

A few constraints

- physically interacting "pool balls"
- must detect some collisions and model their results on the motion
- allow the user to direct 1+ objects
- needs a game goal + be winnable!
The vPool project

Funky Physics is OK... !

Collisions with walls should be handled...

Collisions with other pool balls should be handled...

You need pockets – or some other game objective

A few examples to get you thinking...

So far, VPython has eventually worked for everyone...
Project options...

Design-your own (by 4/17...)

VPool

...VPython

TextID

...Markov

Picobot!

...CFour
For the final project in IST 338, you have a choice of

- one of our pre-scaffolded projects, all available here, or
- designing your own project

Either way, here is a calendar of dates relating to the project:

- **due 4/17**: choice of project - or description of a custom project (one page)
- **on 4/22 and 4/30**: there will be a shorter class (45min or less) to allow room to work on projects or ask questions about them
  - there will also be shorter assignments (small computational challenges) those weeks
- **on 5/6**: project presentations (5-10 minutes on your project, which need not be 100% finished)
- (no class on 5/13)
- **on 5/17**: final project is due
Enjoy the projects!

An unusual variation on VPool

... we certainly do!