Computing to the max

The not-so-subtle art of singling out the best (and worst) of anything...

a comparison comparison

'm+ms' > 'coffee'

[0, 42] > [4, 2]

[0, 'm+ms'] > [4, 'coffee']

> or <

Computing with language

• What's in a Writ1 paper, anyway?
• Battle-tested ciphers & how to break them...

Last hw?

N-step sleepwalking?
Turtle graphics??
Artistic renderings!!!

This week!

Hw #4 due next Monday...

pr0: Are we The Matrix?
pr1: Lab: sounds good...
pr2: Sorting + Caesar!
ex cr: Add'l UIOLI fun'!
areas from rectangles

Areas of 4 rectangles

Areas of 8 rectangles
hw3pr3: areas from rectangles

Area of N rectangles \textit{in the limit}
hw3pr3: Maya Lin, Architect...
Maya Lin, *Artist and Computer Scientist*...

"two-by-four landscape"
Maya Lin, *Artist and Computer Scientist*...

One building block, deliberately applied, *over 50,000 times*...
Building blocks == CS!

`scaledfracs(low,hi,N)`

`f_of_fracs(f,low,hi,N)`

`integrate(f,low,hi,N)`

where are the LCs?

only a few lines...

Areas of 8 rectangles
max

A recipe for life?
The real problem is knowing what we want to maximize!

A recipe for life?
and python already has it for us...
max

A recipe for life?
and python already has it for us...

The real problem is knowing what we want to maximize!

... or minimize, with min
to the \textbf{max}

Want the highest price?

$$\text{max}( [475.5, 458.0, 441.3, 470.8, 532.8, 520.9] )$$

What if the months are in there, as well?

$$\text{max}( [ [470.8,'\text{may'}], [532.8,'\text{jul'}], [520.9,'\text{sep'}] ] )$$

$$\text{max}( [ ['\text{may}',470.8], ['\text{jul}',532.8], ['\text{sep}',520.9] ] )$$
to the max

Want the highest price?

\[
\text{max}( [475.5, 458.0, 441.3, 470.8, 532.8, 520.9] )
\]

What if the months are in there, as well?

\[
\text{max}( [ [470.8, \text{`may'}], [532.8, \text{`jul'}], [520.9, \text{`sep'}] ] )
\]
\[
\text{max}( [ [\text{`may'}, 470.8], [\text{`jul'}, 532.8], [\text{`sep'}, 520.9] ] )
\]

**Mudd's max?**

\[L = [\text{`Harvey'}, \text{`Mudd'}, \text{`College'}, \text{`seeks'}, \text{`to'}, \text{`educate'}, \text{`engineers'}, , \text{`scientists'}, \text{`and'}, \text{`mathematicians'}, \text{`well-versed'}, \text{`in'}, \text{`all'}, \text{`of'}, \text{`these'}, \text{`areas'}, \text{`and'}, \text{`in'}, \text{`the'}, \text{`humanities'}, \text{`and'}, \text{`the'}, \text{`social'}, \text{`sciences'}, \text{`so'}, \text{`that'}, \text{`they'}, \text{`may'}, \text{`assume'}, \text{`leadership'}, \text{`in'}, \text{`their'}, \text{`fields'}, \text{`with'}, \text{`a'}, \text{`clear'}, \text{`understanding'}, \text{`of'}, \text{`the'}, \text{`impact'}, \text{`of'}, \text{`their'}, \text{`work'}, \text{`on'}, \text{`society'}] \]

Or Mudd's min?

\[\text{min}(\text{MSt})\]
def max( L ):
    """ returns the max element from L
    input:  L, a nonempty list
    """
    if len(L) < 2:  return L[0]  # only 1 elem.

    maxOfRest = max(L[1:])  # max of the rest

    if L[0] > maxOfRest:
        return L[0]  # either L[0]
    else:
        return maxOfRest  # or maxOfRest!
```python
def maxSS(L):
    """ returns L's highest scrabble-scoring element (input: L, a nonempty list) """
    if len(L) < 2:
        return L[0]  # only 1 elem.
    maxOfRest = maxSS(L[1:])  # rest's max
    if L[0] > maxOfRest:
        return L[0]  # either L[0]
    else:
        return maxOfRest  # or maxOfRest!
```

L = ['aliens', 'zap', 'hazy', 'code']

Which element has the highest scrabble score? Spacey! I like it!
def maxSS(L):
    """ returns L's highest scrabble-scoring element (input: L, a nonempty list) """

    if len(L) < 2:
        return L[0]  # only 1 elem.

    maxOfRest = maxSS(L[1:])  # rest's max

    if sScore(L[0]) > sScore(maxOfRest):
        return L[0]  # either L[0]
    else:
        return maxOfRest  # or maxOfRest!
```python
def maxSS(L):
    """ returns L's highest scrabble-scoring element (input: L, a nonempty list) """
    if len(L) < 2:
        return L[0]  # only 1 elem.
    maxOfRest = maxSS(L[1:])  # rest's max
    if sScore(L[0]) > sScore(maxOfRest):
        return L[0]  # either L[0]
    else:
        return maxOfRest  # or maxOfRest!
```

Let's see if we can simplify this process... just for LoLs!

L = ['aliens', 'zap', 'hazy', 'code']
Let's see if we can simplify this process... just for LoLs!

Which element has the highest scrabble score?

Spacey! I like it!
A more *comprehensive* solution: \texttt{LoL}

```python
L = [ 'aliens', 'zap', 'hazy', 'code' ]

def maxSS( L ):
    """ returns L's max-scrabble-score word """
    LoL = [ [ sScore(w), w ] for w in L ]

    bestpair = max( LoL )

    return bestpair[1]
```

This does look funny!
A more comprehensive solution

```python
def maxSS(L):
    """ returns L's max-scrabble-score word """
    LoL = [ [sScore(w), w] for w in L ]

    bestpair = max( LoL )

    return bestpair[1]
```

L = ['aliens', 'zap', 'hazy', 'code']
A more comprehensive solution

def maxSS(L):
    """ returns L's max-scrabble-score word """
    LoL = [ [ sScore(w), w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]

This does look funny!

L = ['aliens', 'zap', 'hazy', 'code']
A more comprehensive solution

```python
def maxSS(L):
    """ returns L's max-scrabble-score word ""
    LoL = [[sScore(w), w] for w in L]
    bestpair = max(LoL)
    return bestpair[1]
```

This does look funny!
A more *comprehensive* solution

```python
L = ['aliens', 'zap', 'hazy', 'code']

def maxSS(L):
    """ returns L's max-scrabble-score word """
    LoL = [[sScore(w), w] for w in L]
    bestpair = max(LoL)
    return bestpair[1]
```

I *loathe* hazy code!

L = ['aliens', 'zap', 'hazy', 'code']

LoL = [[6, 'aliens'], [14, 'zap'], [19, 'hazy'], [7, 'code']]

bestpair = max(LoL)

bestpair = [19, 'hazy']

return bestpair[1]

'hazy'
def lastrest(L):
    """ another example – what's returned? """
    LoL = [[w[1:], w] for w in L]
    bestpair = max(LoL)
    return bestpair[1]
def lastrest( L ):
    """ another example — what's returned? """
    LoL = [ [w[1:], w] for w in L ]
    bestpair = max( LoL )
    return bestpair[1]
Everything ... is a max problem?

```python
L = ['aliens', 'zap', 'hazy', 'code']

def lastrest(L):
    """ another example – what's returned? ""
    LoL = [ [w[1:], w] for w in L ]
    LoL = [ [ 'liens', 'aliens'], [ 'ap', 'zap'], [ 'azy', 'hazy'], [ 'ode', 'code'] ]

    bestpair = max( LoL )
    bestpair = [ 'ode', 'code']

    return bestpair[1]
    'code'
```
Everything ... is a max problem?

```python
def lastrevved(L):
    """ another example - what's returned? """
    LoL = [ [w[::-1], w] for w in L ]
    bestpair = max(LoL)
    return bestpair[1]
```

I know the best word here... but does Python?
Everything ... is a max problem?

```python
def lastrevved(L):
    """ another example – what's returned? """
    LoL = [ [w[::-1], w] for w in L ]
    bestpair = max( LoL )
    return bestpair[1]
```

I know the best word here... but does Python?

L = ['aliens', 'zap', 'hazy', 'code']

LoL = [ [sneila, 'aliens'], [paz, 'zap'], [yzah, 'hazy'], [edoc, 'code'] ]

bestpair = [ 'yzah', 'hazy' ]

return 'hazy'
Other examples...

What is `bestnumb`?

```python
>>> bestnumb([10,20,30,40,50,60,70])
40
```

What is `mostnumb`?

```python
>>> mostnumb([100,200,300,400])
100
```

```python
>>> bestnumb([1,2,3,4,5,6,7,8,7])
8
```

```python
>>> mostnumb([1,2,3,4,5,6,7,8,7])
7
```
def maxlen(L):
    LoL = [ [len(s), s] for s in L ]
    bstpr = max(LoL)
    return bstpr[1]

def bestnumb(L):
    """ returns the # in L closest to 42 ""
    LoL = []
    bstpr =
    return bstpr[1]

def mostnumb(L):
    """ returns the item most often in L ""
    LoL = []
    bstpr =
    return bstpr[1]
Quiz

L = [ 'aliens', 'zap', 'hazy', 'code' ]

def maxlen(L):
    LoL = [ [len(s),s] for s in L ]
    bstpr = max(LoL)
    return bstpr[1]

2. What is bstpr?  [6,'aliens']
3. What is returned?  'aliens'

L = [ 30, 40, 50 ]

def bestnumb(L):
    """ returns the # in L closest to 42 """
    LoL = [ [abs(x-42),x] for x in L ]
    bstpr = min(LoL)
    return bstpr[1]

Hint: Python has abs(x) built-in

L = [3,4,5,6,7]

def mostnumb(L):
    """ returns the item most often in L """
    LoL = [ [count(e,L),e] for e in L ]
    bstpr = max(LoL)
    return bstpr[1]

Hint: Use this helper function!

def count(e,L):
    """ return # of e's in L """
    LC = [ 1 for x in L if x == e ]
    return sum(LC)

Extra!

Change exactly three characters in this code so that 3 is returned.
def maxlen(L):
    LoL = [ [len(s),s] for s in L ]

    bstpr = max( LoL )

    return bstpr[1]

Extra!  Change exactly **three** characters in this code so that **3** is returned.
```python
def bestnumb( L ):
    """ returns the # closest to 42 in L """

    LoL = [ [abs(x-42),x] for x in L ]

    bstpr = min( LoL )

    return bstpr[1]
```

```python
def count( e, L ):
    """ returns the # of e's in L """
    LC = [ 1 for x in L if x==e ]
    return sum( LC )

def mostnumb( L ):
    """ returns the item most often in L """
    LoL = [ [count(e,L),e] for e in L ]
    bstpr = max( LoL )
    return bstpr[1]
```

Could you use `x` here instead of `e`?
def count(e, L):
    """ returns the # of e's in L """
    LC = [1 for x in L if x==e]
    return sum(LC)

def mostnumb(L):
    """ returns the item most often in L """
    LoL = [[count(e, L), e] for e in L]
    bstpr = max(LoL)
    return bstpr[1]

Could you use x here instead of e?
Today's lab: *big data*?

Any guesses as to what *kind* of data this is?
Today's lab: *sound* data!

what are the vertical and horizontal axes here?
Lab3 ~ Sound

continuous variation of air pressure vs. time

samples taken every 1/22050th of a second (or some sampling rate)

Each sample is measured on a loudness scale from -32,768 to 32,767. (This fits into 2 bytes.)

These two bytes are called a frame. Raw audio data - such as what is written to the surface of a CD - is simply a list of these frames.

pulse code modulation = PCM data

some examples...

play('swnotry.wav') # run demo()
flipflop('swnotry.wav')
play('swfaith.wav')
changeSpeed( 'swfaith.wav', 44100 )
reverse('swfaith.wav')
play('spam.wav')
reverse('spam.wav')
Lab 4's key challenge...

```python
def flipflop(filename):
    """flipflop swaps the halves of an audio file
    input: filename, the name of the original file
    output: no return value, but
            this creates the sound file 'out.wav'
            and plays it"
    
    print("Playing the original sound...")
    play(filename)

    print("Reading in the sound data...")
    sound_data = [0,0]
    read_wav(filename,sound_data)
    samps = sound_data[0]
    sr = sound_data[1]

    print("Computing new sound...")
    # this gets the midpoint and calls it x
    x = len(samps)//2
    newsamps = samps[x:] + samps[:x]

    newsr = sr
    new_sound_data = [ newsamps, newsr ]

    print("Writing out the new sound data...")
    write_wav( new_sound_data, "out.wav" ) # write data to out.wav

    print("Playing new sound...")
    play( 'out.wav' )
```
Computing with *language*

- ideas / meaning
- language / words / phrases
- strings
- numbers / bits

Python strings are here. "alphabetic processions"
Computing with *language*

- **Ideas / Meaning**: open questions in AI...
- **Language / Words / Phrases**: processing language – *how English-y is it?*
- **Strings**: how strings are represented and stored
- **Numbers / Bits**: Next week...

This week...

Eliza, Siri, Tay ... trouble?
Caesar Cipher: \texttt{encipher}

\texttt{encipher(s, n)}

- \texttt{encipher( 'I <3 Latin' , 0 )} \rightarrow 'I <3 Latin'
- \texttt{encipher( 'I <3 Latin' , 1 )} \rightarrow 'J <3 Mbujo'
- \texttt{encipher( 'I <3 Latin' , 2 )} \rightarrow 'K <3 Ncvkp'
- \texttt{encipher( 'I <3 Latin' , 3 )} \rightarrow 'L <3 Odwlq'
- \texttt{encipher( 'I <3 Latin' , 4 )} \rightarrow 'M <3 Pexmr'
- \texttt{encipher( 'I <3 Latin' , 5 )} \rightarrow 'N <3 Qfyns'
- \texttt{encipher( 'I <3 Latin' , 25 )} \rightarrow 'H <3 Kzshm'

What is it doing?
Caesar Cipher: `encipher`

`encipher(s, n)` should return the string `s` with each *alphabetic* character shifted/wrapped by `n` places in the alphabet.

- `encipher( 'I <3 Latin' , 0 )` returns 'I <3 Latin'
- `encipher( 'I <3 Latin' , 1 )` returns 'J <3 Mbujo'
- `encipher( 'I <3 Latin' , 2 )` returns 'K <3 Ncvkp'
- `encipher( 'I <3 Latin' , 3 )` returns 'L <3 Odwlq'
- `encipher( 'I <3 Latin' , 4 )` returns 'M <3 Pexmr'
- `encipher( 'I <3 Latin' , 5 )` returns 'N <3 Qfyns'
- ...
- `encipher( 'I <3 Latin' , 25 )` returns 'H <3 Kzshm'
The SAME bits represent an integer or a string, depending on type: int or str
Unicode

Universal Character Encoding

Some fun characters...

chr(156265)  chr(9835)  chr(9731)

My favorite is chr(1661)

u"I <3 "u202e Wow! "u202d Unicode..."

on Win10: chcp 65001

http://ian-albert.com/unicode_chart/
The SAME bits represent an integer or a string, depending on type: `int` or `str`.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010 1111</td>
<td>47</td>
<td>2F</td>
<td>/</td>
</tr>
<tr>
<td>0011 0000</td>
<td>48</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>0011 0001</td>
<td>49</td>
<td>31</td>
<td>1</td>
</tr>
</tbody>
</table>

Identical bits are stored in each variable!

The types determine how to interpret the bits; the names don’t matter at all…

1 byte
8 bits

over in memory…
ASCII and Unicode

This is why 'CS' < 'clear'!
This is why 'CS' < 'clear'!
rot13('a') should output 'n'
rot13('M') should output 'Z'
rot13('n') should output 'a'
rot13('W') should output 'J'
rot13(' ') should output ' '  
rot13('<>') should output '<'
## ASCII and Unicode

<table>
<thead>
<tr>
<th>chr value</th>
<th>ord value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcdefghijklmnopqrstuvwxyz</td>
<td>97 99 101 103 105 107 109 111 113 115 117 119 122</td>
</tr>
<tr>
<td>ABCDEFGHIJKLMNOPQRSTUVWXYZ</td>
<td>65 67 69 71 73 75 77 79 81 83 85 87 90</td>
</tr>
</tbody>
</table>

### Questions

- What is `ord('U') // 2`?
- What is `chr(ord('i') + 13)`?
- What is `chr(ord('W') + 13)`?

---

convert # to char

<table>
<thead>
<tr>
<th>chr</th>
<th>ord</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>chr</th>
<th>ord</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

convert char to #

- how do we wrap?
def rot13( c ):  
    """ rotates c by 13 chars, "wrapping" as needed
    NON-LETTERS don't change!
    """
    if 'a' <= c <= 'z':
        if ord(c)+13 <= ord('z'):
            return chr( ord(c)+13 )
        else:
            return chr(  )
    elif 'A' <= c <= 'Z':  
        return chr(  )
    else:
        return chr(  )

Extra: How would you rotate n places, instead of 13?
def rot13( c ):
    """ rotates c by 13 chars, "wrapping" as needed
    NON-LETTERS don't change!
    """
    if 'a' <= c <= 'z':
        if ord(c)+13 <= ord('z'):
            return chr( ord(c)+13 )
        else:
            return chr( ord(c)+13-26 )
    elif 'A' <= c <= 'Z': # upper-case test!
        same, but using 'Z'
    else:
        return c

(0) What do these tests do?
(1) What code will "wrap" to the alphabet's other side?
(2) How will upper case change? Try noting only the code differences...
(3) What if c is not a letter at all?

Extra: How would you rotate n places, instead of 13?
rot13's surprising history...
Caesar Cipher: `encipher`

```python
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.','25)
'Aycqyp agnfcp? G npcdcp Aycqyp qyjyb.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.','15)
'Qosgof qwdvsf? W dfstsf Qosgof gozor.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.','4)
'Fdhvdu flskhu? L suhihu Fdhvdu vdodg.'

>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.','1)
'Caesar cipher? I prefer Caesar salad.'

>>> encipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvynla \\
   'lclyfaopun dl ohcl slhyulk.','19)
'An education is what remains after we forget everything we have learned.'
```
Caesar Cipher: `decipher`

```python
>>> decipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc."
'Caesar cipher? I prefer Caesar salad."
```

```python
>>> decipher('Hu lkbjhapvu pz doha ylthpuz hmaly dl mvynla \'
lclyfaopun dl ohcl slhyulk."
'An education is what remains after we forget everything we have learned."
```

```python
>>> decipher('Uifz xpsl ju pvu xjui b qfodjm!"
```

```python
>>> decipher('gv vw dtwvg"
```
Decipher?

Strategies?

*Algorithms*?
Decipher?

All possible decipherings

Strategies?

Algorithms?
Decipher?

Strategies?

Algorithms?

All possible decipherings

Score them all

quantifying Englishness?
Decipher?

All possible decipherings

[0, 'gv vw dtwvg'],
[2, 'hw wx euwh'],
[2, 'ix xy fvyxi'],
[0, 'jy yz gwzyj'],
[2, 'kz za hxazk'],
[0, 'la ab iybal'],
[2, 'mb bc jzcbm'],
[4, 'nc cd kadc'],
[3, 'pe ef mcfep'],
[0, 'qf fg ndgf'],
[2, 'rg gh oehg'],
[2, 'sh hi pfih'],
[3, 'ti ij qgj'],
[2, 'uj jk rhkJ'],
[1, 'vk kl silk'],
[0, 'wl lm tjml'],
[1, 'xn mn uknm'],
[2, 'yn no vl'],
[3, 'zo op wmpoz'],
[2, 'ap pq xnqpa'],
[1, 'bq qr yorq'],
[0, 'cr rs zp'],
[1, 'ds st aqtd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']

Max!

Score them all

yields the "most English" phrase"
Measuring *Englishness*

**Very English-y**

- "Call me Ishmael."
- "Attack at dawn!"
- "rainbow, table, candle"
- "Yow! Legally-imposed CULTURE-reduction is CABBAGE-BRAINED!"
- "quadruplicity drinks procrastination"
- "Hold the newsreader's nose squarely, waiter, or friendly milk will countermand my trousers."
- "the gostak distims the doshes"
- "hension, framble, bardle"
- "jufict, stofwus, lictpub"
- "itehbs, rsnevtr, khbsota"
- "epadxo, nojarpn, gdxokpw"
- "h o q dedqBzdrzqrzkzc"

**Not English-y**

higher scores

lower scores

quantifying Englishness?

All of these sound good to me!
Decipher?

Strategies?

Algorithms?

All possible decipherings:

- `gv vv dtwvg`
- `hw wx euxwh`
- `ix xy fvyxi`
- `jy yz gwzyj`
- `kz za hxazk`
- `la ab iybal`
- `mb bc jzcbo`
- `nc cd kadcno`
- `od de lbdeto`
- `pe ef mcfep`
- `qf fg ndgfq`
- `rg gh oehgr`
- `sh hi pfishi`
- `ti ij qgjiti`
- `uj jk rhkaju`
- `vk kl silkv`
- `wl lm tjmlw`
- `xm mn uknmx`
- `yn no vlony`
- `zo op wmpoz`
- `ap pq xnpqa`
- `bq qr yorqb`
- `cr rs zpsrc`
- `ds st aqtsd`
- `et tu brute`
- `fu uv csvuf`

Score them all:

Max:

- `[4, 'la ab iybal']`
- `[0, 'mb bc jzcbo']`
- `[1, 'nc cd kadcno']`
- `[4, 'od de lbdeto']`
- `[3, 'pe ef mcfep']`
- `[0, 'qf fg ndgfq']`
- `[2, 'rg gh oehgr']`
- `[2, 'sh hi pfishi']`
- `[3, 'ti ij qgjiti']`
- `[2, 'uj jk rhkaju']`
- `[1, 'vk kl silkv']`
- `[0, 'wl lm tjmlw']`
- `[1, 'xm mn uknmx']`
- `[2, 'yn no vlony']`
- `[0, 'zo op wmpoz']`
- `[1, 'ap pq xnpqa']`
- `[1, 'bq qr yorqb']`
- `[0, 'cr rs zpsrc']`
- `[1, 'ds st aqtsd']`
- `[4, 'et tu brute']`
- `[3, 'fu uv csvuf']`

"Englishness" score based on # of vowels.
### Decipher?

<table>
<thead>
<tr>
<th>Score</th>
<th>Deciphering</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.9e-05</td>
<td>'gv vw dtwvg'</td>
</tr>
<tr>
<td>3.6e-05</td>
<td>'hw wx euxwh'</td>
</tr>
<tr>
<td>1.4e-07</td>
<td>'ix xy fvyxi'</td>
</tr>
<tr>
<td>8.8e-11</td>
<td>'jy yz gwzyj'</td>
</tr>
<tr>
<td>7.2e-10</td>
<td>'kz za hxazk'</td>
</tr>
<tr>
<td>0.01503</td>
<td>'la ab iybal'</td>
</tr>
<tr>
<td>3.7e-08</td>
<td>'mb bc jzcbm'</td>
</tr>
<tr>
<td>0.00524</td>
<td>'nc cd kadcn'</td>
</tr>
<tr>
<td>0.29041</td>
<td>'od de lbedo'</td>
</tr>
<tr>
<td>0.00874</td>
<td>'pe ef mcfep'</td>
</tr>
<tr>
<td>7.3e-07</td>
<td>'qf fg ndgfq'</td>
</tr>
<tr>
<td>0.06410</td>
<td>'rg gh oehgr'</td>
</tr>
<tr>
<td>0.11955</td>
<td>'sh hi pfihs'</td>
</tr>
<tr>
<td>3.1e-06</td>
<td>'ti ij qgjit'</td>
</tr>
<tr>
<td>1.1e-08</td>
<td>'uj jk rhkju'</td>
</tr>
<tr>
<td>2.6e-07</td>
<td>'vk kl silkv'</td>
</tr>
<tr>
<td>5.7e-08</td>
<td>'ap pq xnqpa'</td>
</tr>
<tr>
<td>0.00024</td>
<td>'cr rs zpsrc'</td>
</tr>
<tr>
<td>0.00060</td>
<td>'ds st aqtst'</td>
</tr>
<tr>
<td>0.45555</td>
<td>'et tu brute'</td>
</tr>
<tr>
<td>0.00011</td>
<td>'fu uv csvuf'</td>
</tr>
</tbody>
</table>

### Strategies?

- Algorithms?
- "Englishness" based on letter probabilities

---

Decipher?

Decipher?

Decipher?

Decipher?
We'll see you in Lab4!

Earbuds are helpful for lab - unless you *really* like Darth Vader!