cs 5 today

This would make me hungry…
but I ate breakfast this morning!

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 #3 due next Monday…

pr0: Are we *The Matrix?*

pr1: Lab: *sounds good…*

pr2: Sorting + Caesar!

ex cr: Add'l UIOLI *fun'!*

max

A recipe for life ?

Where is this!?

`max`

A recipe for life ?

and python already has it for us…

The real problem is knowing ***<u>what</u>*** we want to maximize!

`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 **max**

And I thought $54 was overpriced!
1,232.41 USD Sep 18, 2019
NASDAQ: GOOG
1,500
1,000
500
0
2007
2011
2015
2019

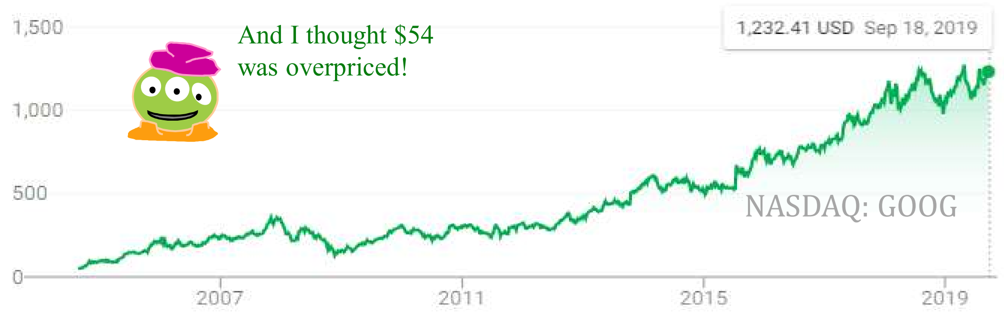
Want the highest price?

```
max( [475.5, 458.0, 441.3, 470.8, 532.8, 520.9] )        ST
       'nov'   'jan'   'mar'   'may'   'jul'   'sep'
```

What if the months are in there, as well?

```
max( [ [470.8,'may'], [532.8,'jul'], [520.9,'sep'] ] )    STm

max( [ ['may',470.8], ['jul',532.8], ['sep',520.9] ] )    mST
```

to the **max**



Want the highest price?

```
max( [475.5, 458.0, 441.3, 470.8, 532.8, 520.9] )
      'nov'    'jan'    'mar'    'may'    'jul'    'sep'
```

ST

What if the months are in there, as well?

```
max( [ [470.8,'may'], [532.8,'jul'], [520.9,'sep'] ] )
```

STm

```
max( [ ['may',470.8], ['jul',532.8], ['sep',520.9] ] )
```

mST

Mudd's **max**?

MSt

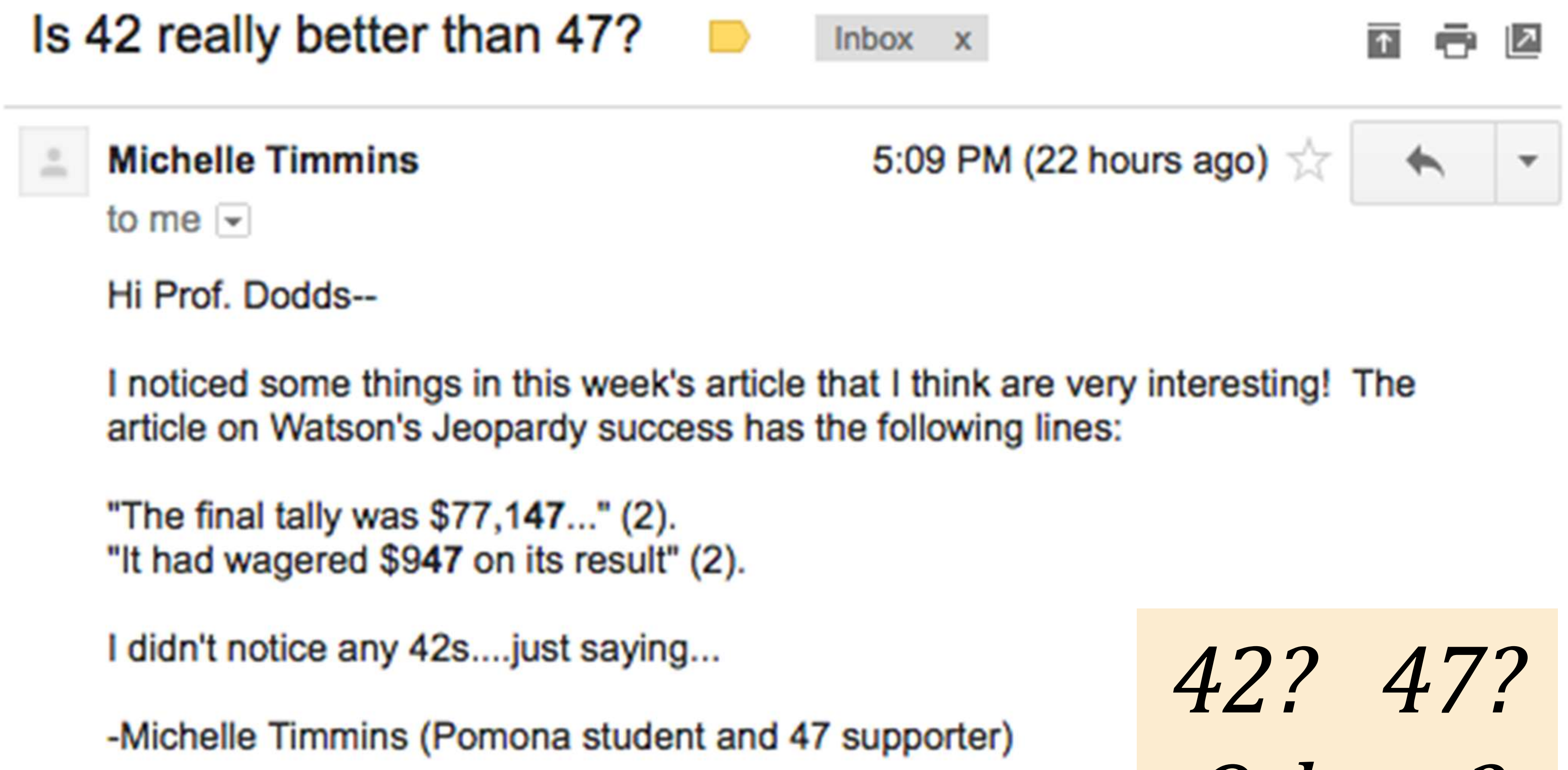
```
L = ['Harvey', 'Mudd', 'College', 'seeks', 'to', 'educate', 'engineers,', 'scientists',
'and', 'mathematicians', 'well-versed', 'in', 'all', 'of', 'these', 'areas', 'and',
'in', 'the', 'humanities', 'and', 'the', 'social', 'sciences', 'so', 'that', 'they',
'may', 'assume', 'leadership', 'in', 'their',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

```
max(MSt)
```

Or Mudd's **min**?

```
min(MSt)
```

'CS' < 'clear'



Is 42 really better than 47?

Michelle Timmins 5:09 PM (22 hours ago)

to me

Hi Prof. Dodds--

I noticed some things in this week's article that I think are very interesting! The article on Watson's Jeopardy success has the following lines:

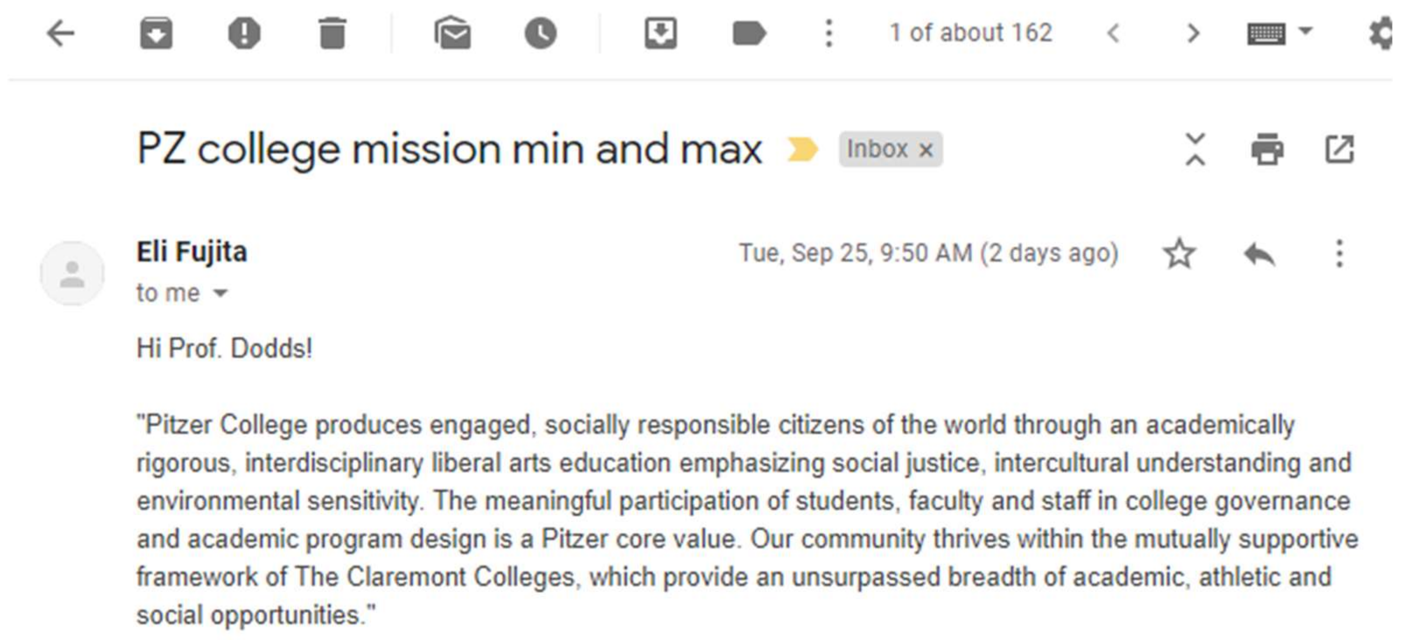
"The final tally was $77,**147**..." (2).
"It had wagered $9**47** on its result" (2).

I didn't notice any 42s....just saying...

-Michelle Timmins (Pomona student and 47 supporter)

42? 47? Others?

`min` + `max`? *Thanks, Eli!*



PZ college mission min and max Inbox ×

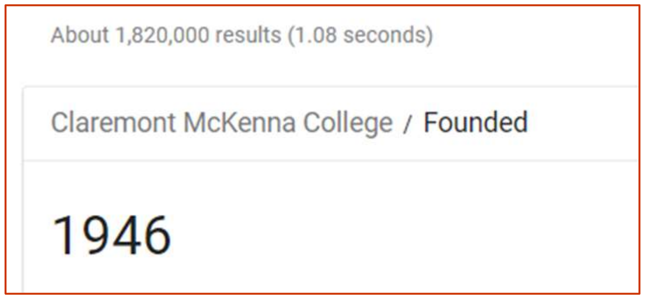
Eli Fujita
to me

Tue, Sep 25, 9:50 AM (2 days ago)

Hi Prof. Dodds!

"Pitzer College produces engaged, socially responsible citizens of the world through an academically rigorous, interdisciplinary liberal arts education emphasizing social justice, intercultural understanding and environmental sensitivity. The meaningful participation of students, faculty and staff in college governance and academic program design is a Pitzer core value. Our community thrives within the mutually supportive framework of The Claremont Colleges, which provide an unsurpassed breadth of academic, athletic and social opportunities."

CMC 40's #: ***46***


About 1,820,000 results (1.08 seconds)

Claremont McKenna College / Founded

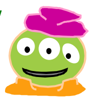
1946

recursive `max`

$$\overbrace{[\ 7,\ \underbrace{10,\ -2,\ 42,\ 15}_{\texttt{L[1:]}}\ ]}^{\texttt{L}}$$

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

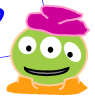
```
def max( L ):
    """ returns the max element from L
        input:  L, a nonempty list
    """
    if len(L) < 2:  return L[0] # only 1 elem.

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

    if          L[0]   >           maxOfRest :
        return L[0]                  # either L[0]
    else:
        return maxOfRest             # or maxOfRest!
```



`max` *with scrabble-score*

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

Which element has the highest scrabble score?



```
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!
```

Spacey! I like it!

max *with scrabble-score*

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

Which element has the highest scrabble score?

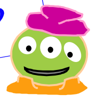
```
def maxSS( L ):
    """ returns L's highest scrabble-scoring
          element (input:  L, a nonempty list)
    """
    if len(L) < 2:  return L[0] # only 1 elem.

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

    if  sScore(L[0]) > sScore(maxOfRest):
        return L[0]                 # either L[0]
    else:
        return maxOfRest            # or maxOfRest!
```

Spacey! I like it!

`max` *with scrabble-score*

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

Which element has the highest scrabble score?



```
def maxSS( L ):
    """ returns L's hig          ble-scoring
                                 onempty list)

                return L[0]  # only 1 elem.

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

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

```
    if  sScore(L[0]) > sScore(maxOfRest):
        return L[0]                # either L[0]
    else:
        return maxOfRest           # or maxOfRest!
```

Spacey! I like it!

A more *comprehensive* solution: **LoL**

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]
```

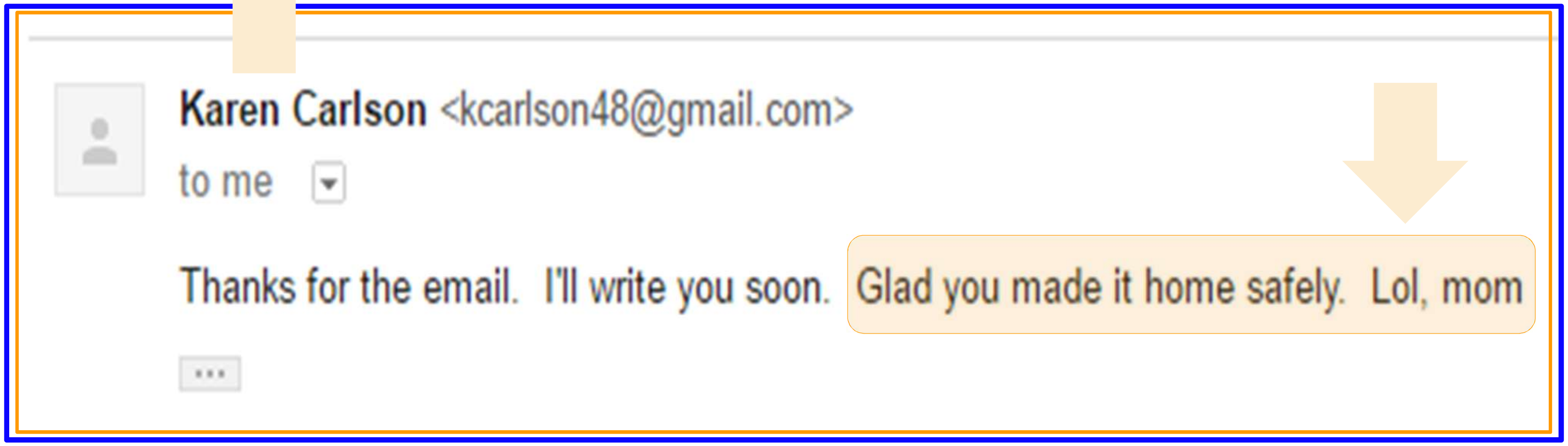
This *does* look funny!

A more *comprehensive* solution

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

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

This *does* look funny!



Karen Carlson <kcarlson48@gmail.com>
to me

Thanks for the email. I'll write you soon. Glad you made it home safely. Lol, mom

A more ***comprehensive*** solution

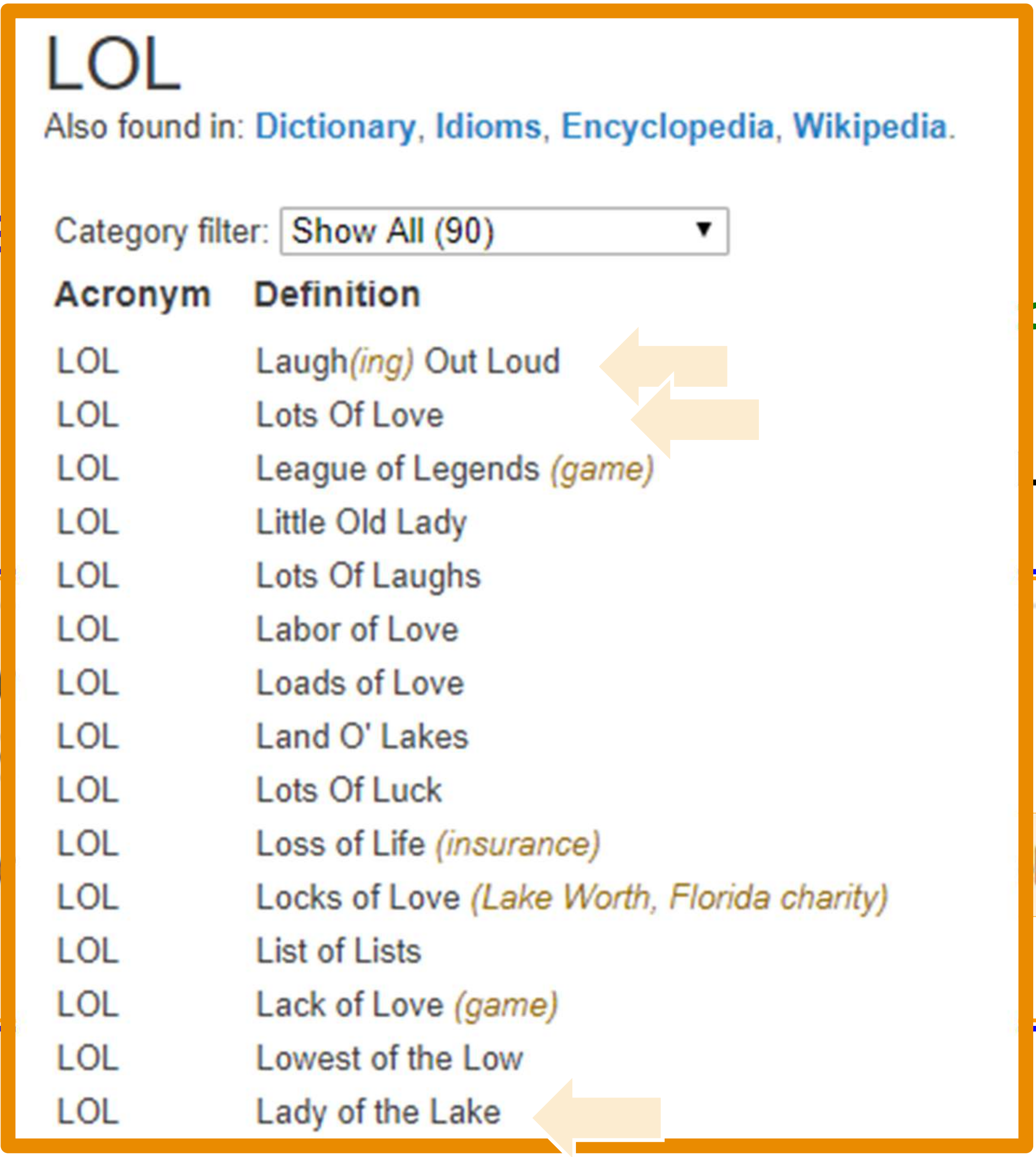
```
def maxS
    """                                                    e word
    """
    LoL                                                    L ]
```

This *does* look funny!

Karen Ca
to me

Thanks fo

ly. Lol, mom


LOL

Also found in: Dictionary, Idioms, Encyclopedia, Wikipedia.

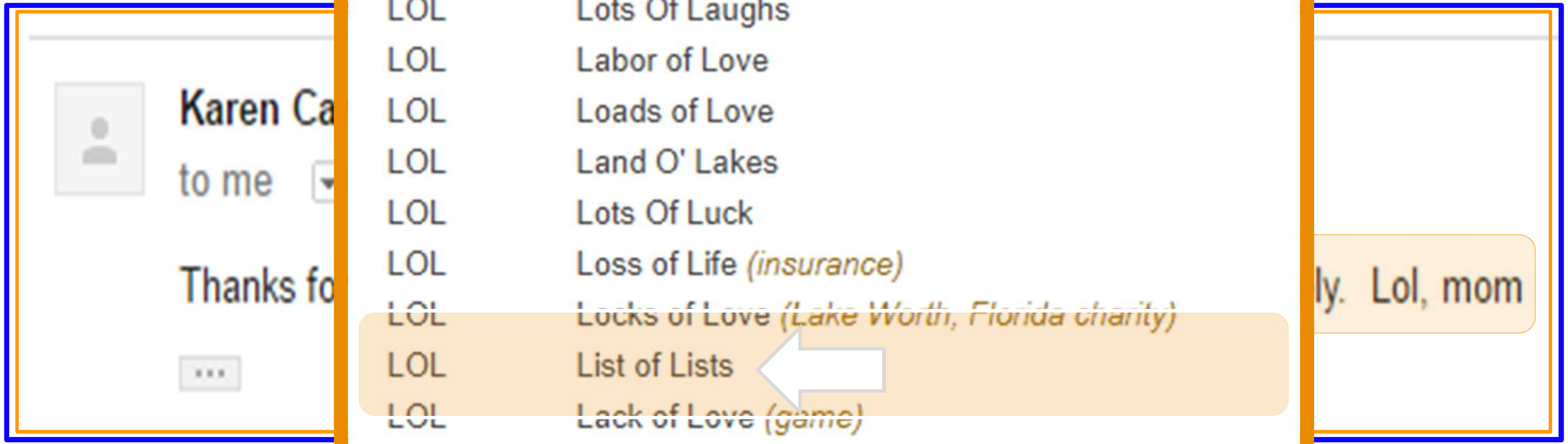
Category filter: Show All (90)

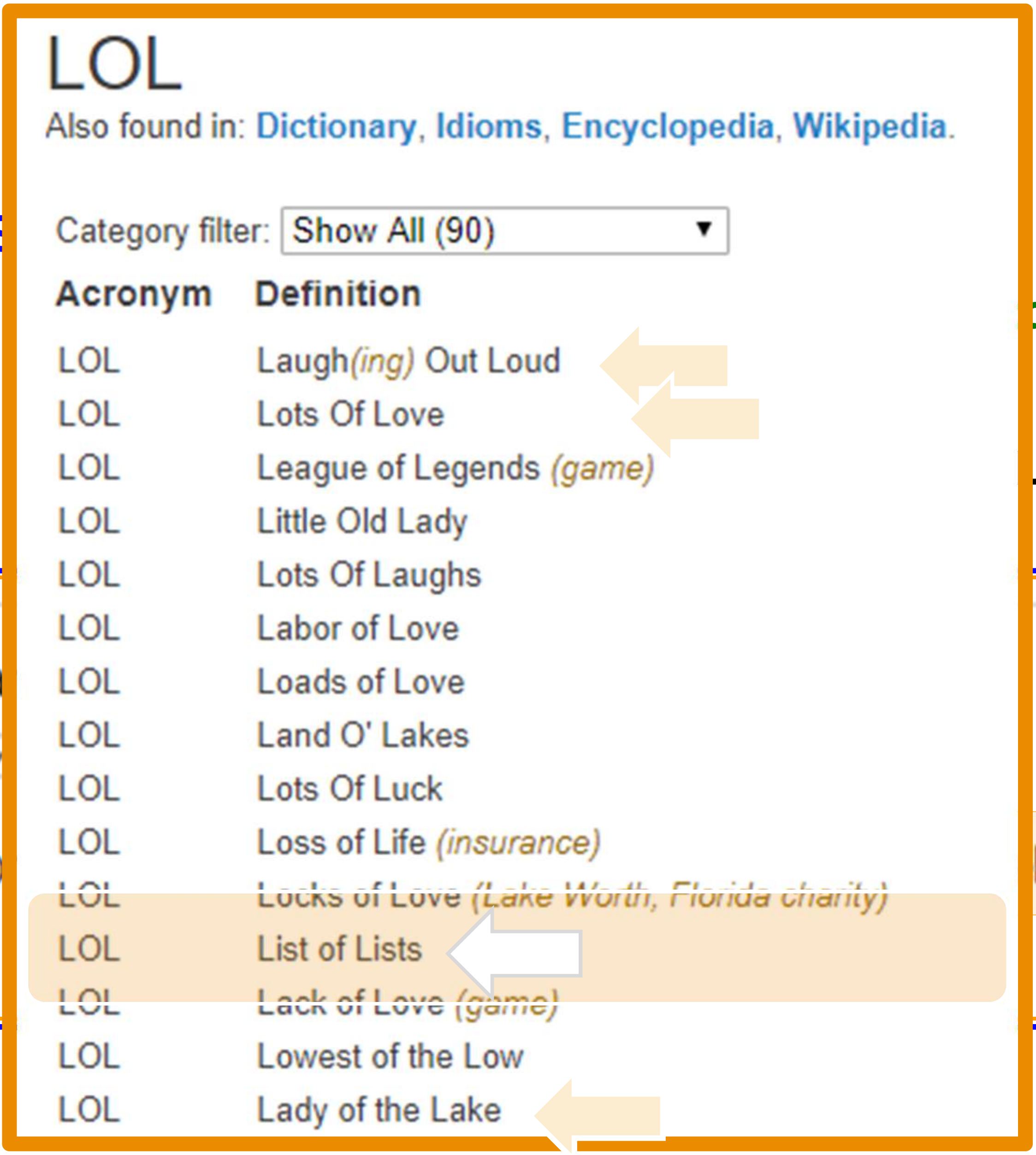
Acronym	Definition
LOL	Laugh*(ing)* Out Loud
LOL	Lots Of Love
LOL	League of Legends *(game)*
LOL	Little Old Lady
LOL	Lots Of Laughs
LOL	Labor of Love
LOL	Loads of Love
LOL	Land O' Lakes
LOL	Lots Of Luck
LOL	Loss of Life *(insurance)*
LOL	Locks of Love *(Lake Worth, Florida charity)*
LOL	List of Lists
LOL	Lack of Love *(game)*
LOL	Lowest of the Low
LOL	Lady of the Lake

A more *comprehensive* solution

```
def maxS
    """                                              e word
    """
    LoL                                              L  ]
```

This *does* look funny!


Karen Ca
to me
Thanks fo
ly. Lol, mom


LOL

Also found in: Dictionary, Idioms, Encyclopedia, Wikipedia.

Category filter: Show All (90)

Acronym	Definition
LOL	Laugh*(ing)* Out Loud
LOL	Lots Of Love
LOL	League of Legends *(game)*
LOL	Little Old Lady
LOL	Lots Of Laughs
LOL	Labor of Love
LOL	Loads of Love
LOL	Land O' Lakes
LOL	Lots Of Luck
LOL	Loss of Life *(insurance)*
LOL	Locks of Love *(Lake Worth, Florida charity)*
LOL	List of Lists
LOL	Lack of Love *(game)*
LOL	Lowest of the Low
LOL	Lady of the Lake

A more ***comprehensive*** solution

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

I ***loathe*** hazy code!

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

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

```
    bestpair = max( LoL )
```

bestpair = [19,'hazy']

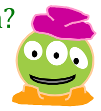
```
    return bestpair[1]
```

'hazy'

Everything ... is a max problem?

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

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



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

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

```
    bestpair = max( LoL )
```

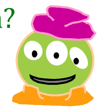
bestpair =

```
    return bestpair[1]
```

Everything ... is a max problem?

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

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



```
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 =

```
    return bestpair[1]
```

Everything ... is a max problem?

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

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

```
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 )
```

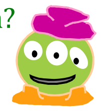
bestpair = ['ode' ,'code']

```
    return bestpair[1]
```

Everything ... is a max problem?

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

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



```
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']

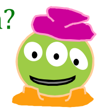
```
    return bestpair[1]
```

'code'

Everything ... is a max problem?

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

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



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

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

```
    bestpair = max( LoL )
```

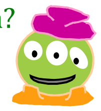
bestpair =

```
    return bestpair[1]
```

Everything ... is a max problem?

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

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



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

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

```
    bestpair = max( LoL )
```

bestpair =

```
    return bestpair[1]
```

Everything ... is a max problem?

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

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

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

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

```
    bestpair = max( LoL )
```

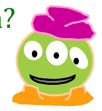
bestpair = ['yzah' ,'hazy']

```
    return bestpair[1]
```

Everything ... is a max problem?

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

I know the best word here… but does Python?



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

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

```
    bestpair = max( LoL )
```

bestpair = ['yzah' ,'hazy']

```
    return bestpair[1]
```

'hazy'

Other examples…

What is **bestnumb** ?

What is **mostnumb** ?

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

>>> bestnumb( [100,200,300,400] )
100

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

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

These functions *have* made me number

Name(s) ____________________

Quiz

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

```
def maxlen(L):
    LoL = [ [len(s),s] for s in L ]
```

1. What is **LoL**? here is a start: LoL is [[6,'aliens'], [3,'zap'], __________, __________]

```
    bstpr = max( LoL )
```

2. What is **bstpr**?

```
    return bstpr[1]
```

3. What is returned?

Extra!

Change exactly ***three*** characters in this code so that **3** is returned.

L = [30, 40, 50]

Use the **LoL** method to write these two functions

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

Hint: Python has **abs(x)** built-in

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

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

Quiz

Try this on the back page first!

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

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

1. What is **LoL**? [[6,'aliens'], [3,'zap'], [4,'hazy'], [4,'code']]

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

3. What is returned? 'aliens'

Extra!

Change exactly ***three*** characters in this code so that **3** is returned.

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

```
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)
```

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

```
def maxlen(L):
    LoL = [ [len(s),s] for s in L ]
```

1. What is `LoL`? [[6,'aliens'], [3,'zap'], [4,'hazy'], [4,'code']]

```
    bstpr = max( LoL )
```

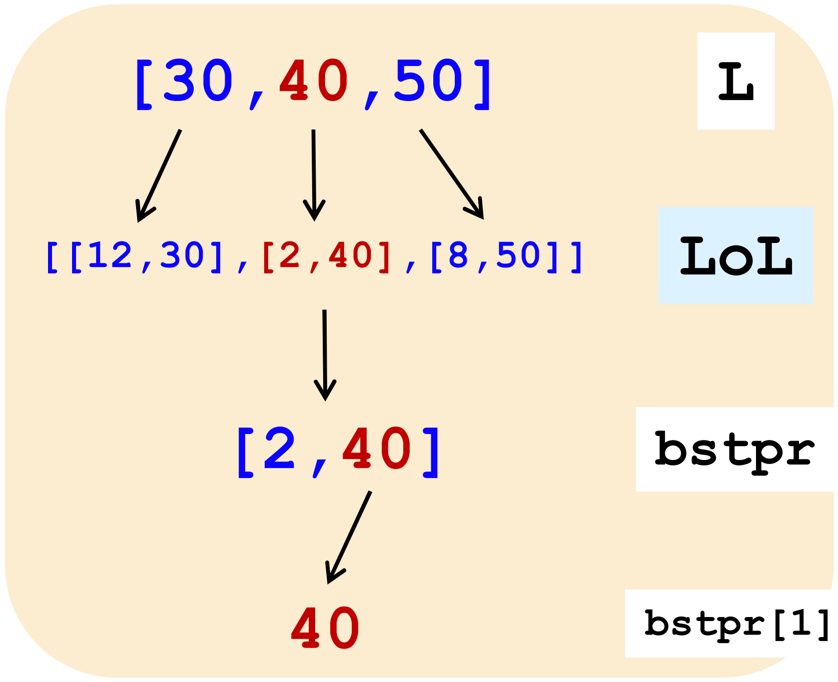
2. What is `bstpr`? [6,'aliens']

```
    return bstpr[1]
```

3. What is returned? 'aliens'

Extra! Change exactly ***three*** characters in this code so that **3** is returned.

bestnumb



[30, 40, 50]

```
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]
```

mostnumb

Helper function: `count(e,L)`

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

$$\begin{tikzcd}[column sep=small]
\texttt{L} & {[6,} \arrow[d] & {7,} \arrow[d] & {7,} \arrow[d] & {8]} \arrow[d] \\
\texttt{LoL} & {[\,[1,6],} & {[2,7],} \arrow[d] & {[2,7],} & {[1,8]\,]} \\
\texttt{bstpr} & & {[2,7]} \arrow[d] & & \\
\texttt{bstpr[1]} & & 7 & &
\end{tikzcd}$$

[6,7,7,8]

```
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]
```

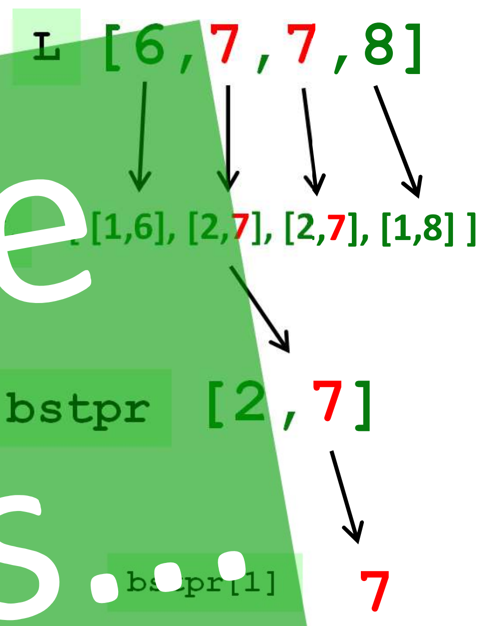
Could you use x here instead of e?

mostnumb

Helper function: `count(e,L)`

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

Pass those backwards...

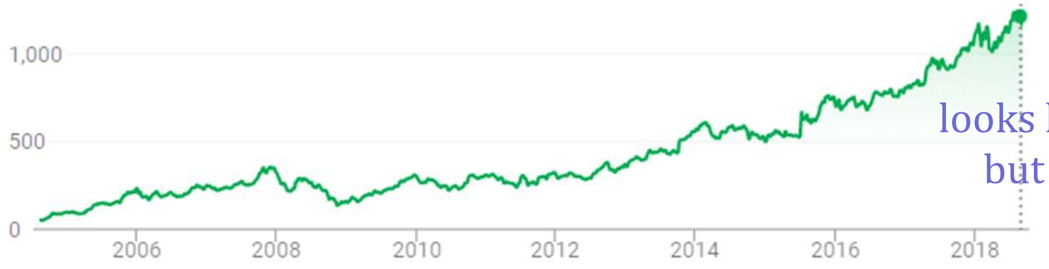


[6,7,7,8]

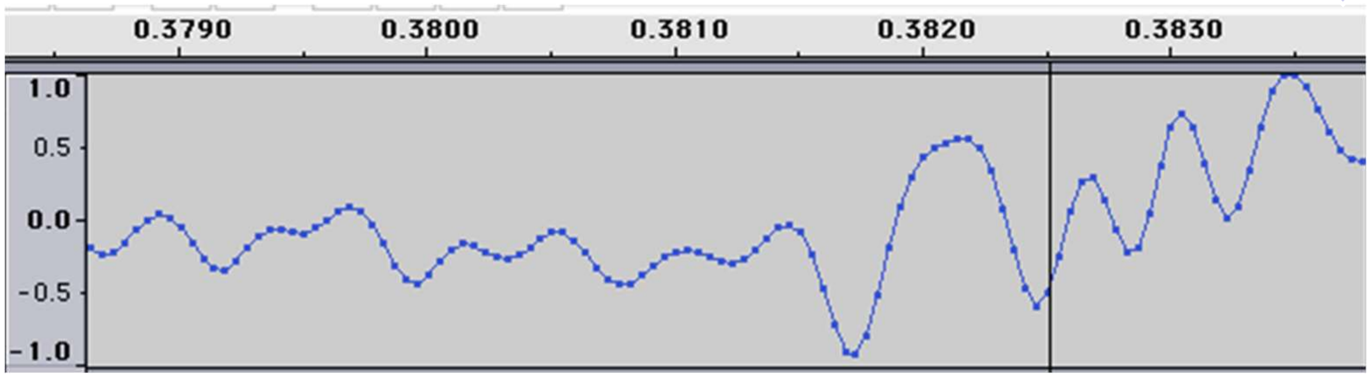
```
def mostnumb( L ):
    """ returns the item most often in L """
    LoL = [ [count(e,L),e] for e in L ]
    bstpr = max( LoL )
    return bstpr[1]
```

Could you use x here instead of e?

Today's lab: *big data?*

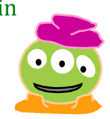


looks like stock prices,
but much wavier!

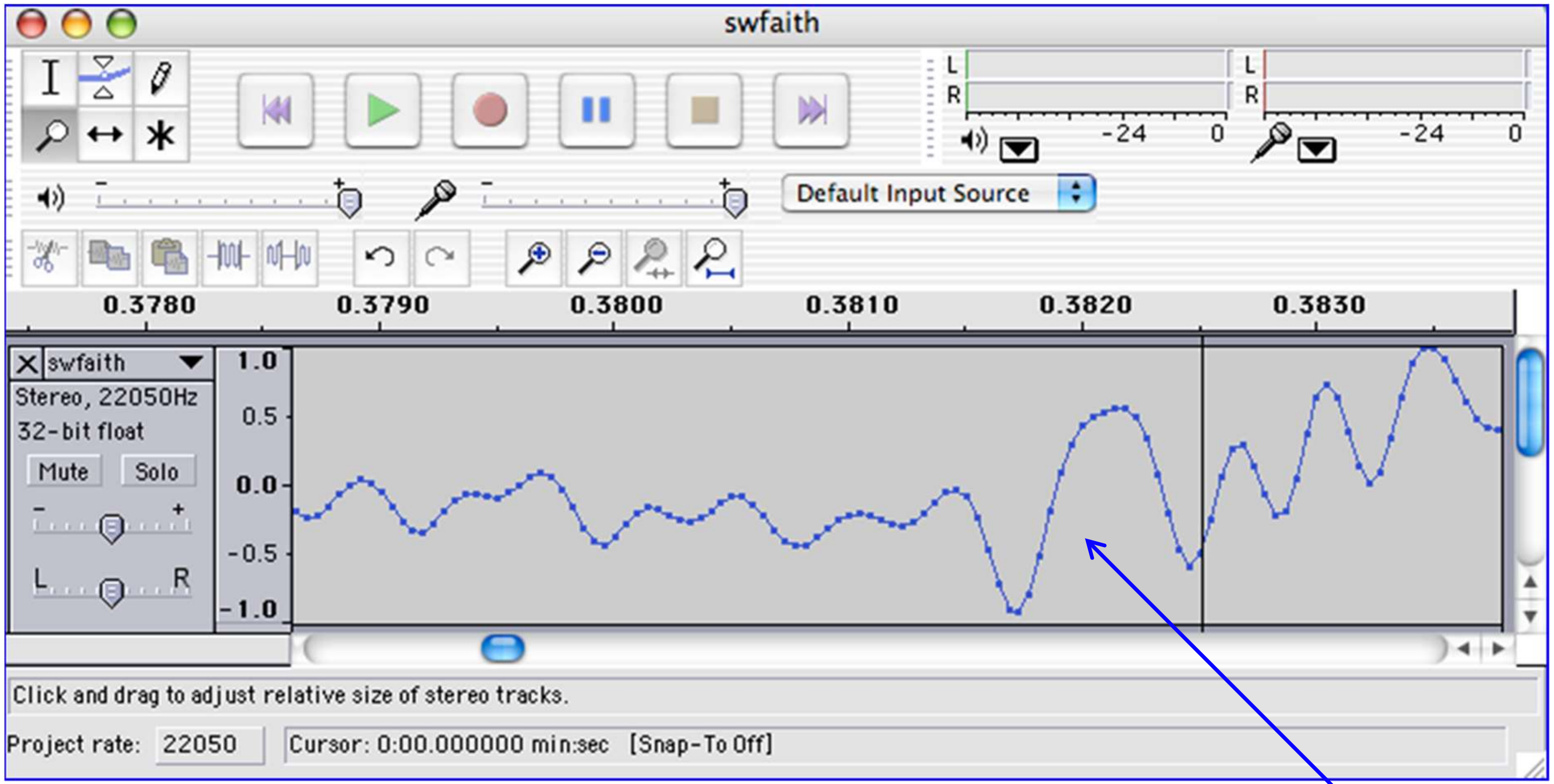


Any guesses as to what ***kind*** of data this is?

I find your lack of faith in
this data disturbing.



Today's lab: *sound* data!



Default Input Source

swfaith
Stereo, 22050Hz
32-bit float

Click and drag to adjust relative size of stereo tracks.

Project rate: 22050

Cursor: 0:00.000000 min:sec [Snap-To Off]

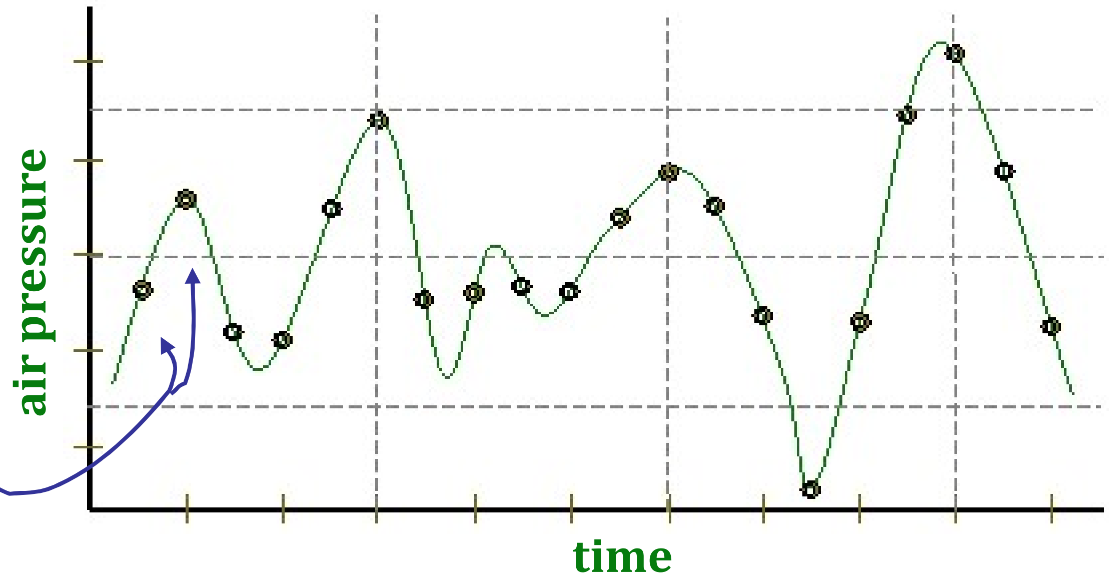
what are the vertical and horizontal axes here?

Lab3 ~ Sound

in **`.wav`** files

physics

continuous variation of air pressure vs. time



sampling

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

quantization

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

storage

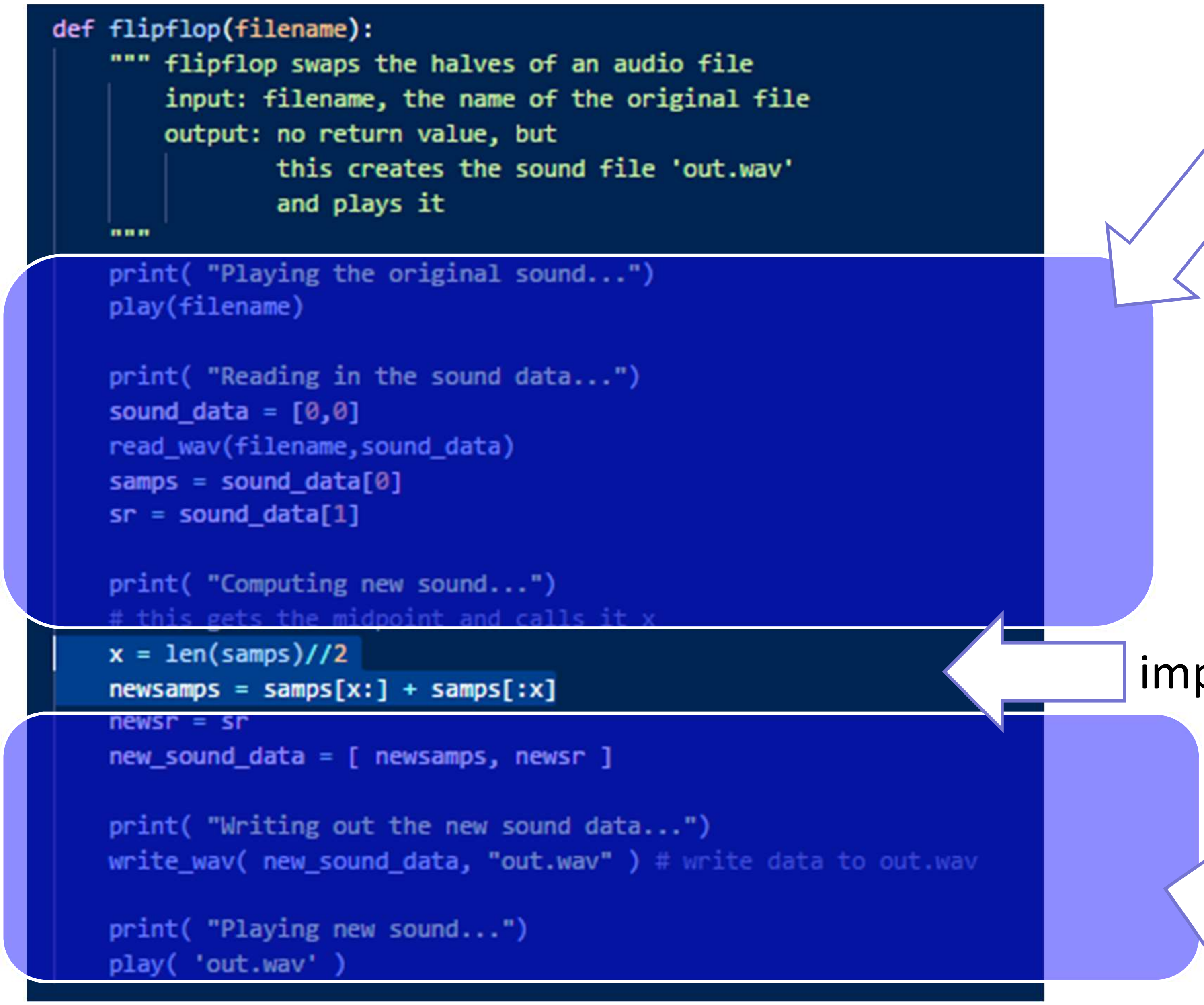
These two bytes are called a *frame*. Raw audio data - such as what is written to the surface of a CD - is simply a list of these frames.

pulse code modulation = PCM data

```
play('swnotry.wav')   # run demo()
flipflop('swnotry.wav')
play('swfaith.wav')
changeSpeed( 'swfaith.wav', 44100 )
reverse('swfaith.wav')
play('spam.wav')
reverse('spam.wav')
```

some examples...

Lab 3'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' )
```

intro stuff – not important

important stuff

"outro" stuff

Computing with *language*

$\longrightarrow$ **ideas / meaning**

$\longrightarrow$ **language / words / phrases**

$\longrightarrow$ **strings**

Python strings are here.

"alphabetic processions"

$\longrightarrow$ **numbers / bits**

Computing with *language*

→ **ideas / meaning**

open questions in AI ...

Eliza, Siri, Tay ... trouble?

→ **language / words / phrases**

This week...

processing language – ***how English-y is it***?

→ **strings**

how strings are represented and stored

Next week...

→ **numbers / bits**

Caesar Cipher: encipher + decipher

```
encipher(s,n)
```

What is it doing?

```
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'
```

Caesar Cipher: encipher + decipher

`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'
```

CA

Caesar

Caesar Cipher: `encipher`

Brutus

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

S1

```
>>> 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.'
```

S2

Caesar

Caesar Cipher: decipher

Brutus

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

S1

S2

```
>>> 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.'
```

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

PL

```
>>> decipher('gv vw dtwvg')
```

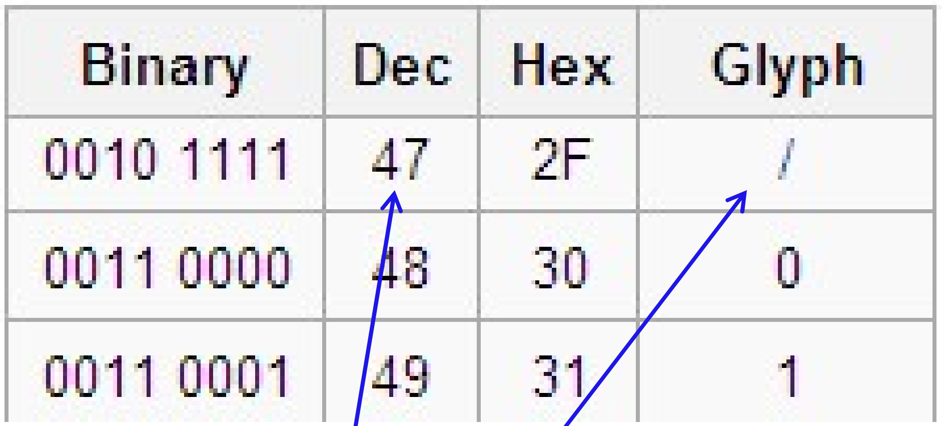
LAT

How!?

Which is more difficult computationally?

ASCII

American Standard Code for Information Interchange



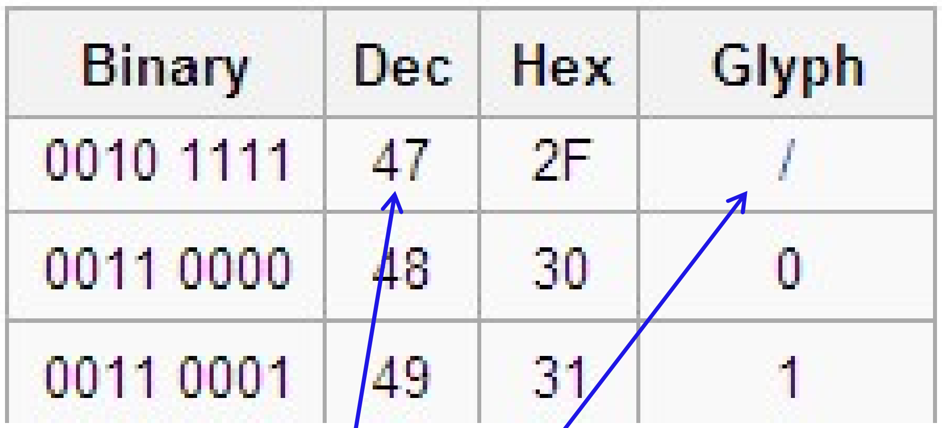
Binary	Dec	Hex	Glyph
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1

1 byte

8 bits

The SAME bits represent an integer or a string, depending on type: `int` or `str`

ASCII *or* Unicode...



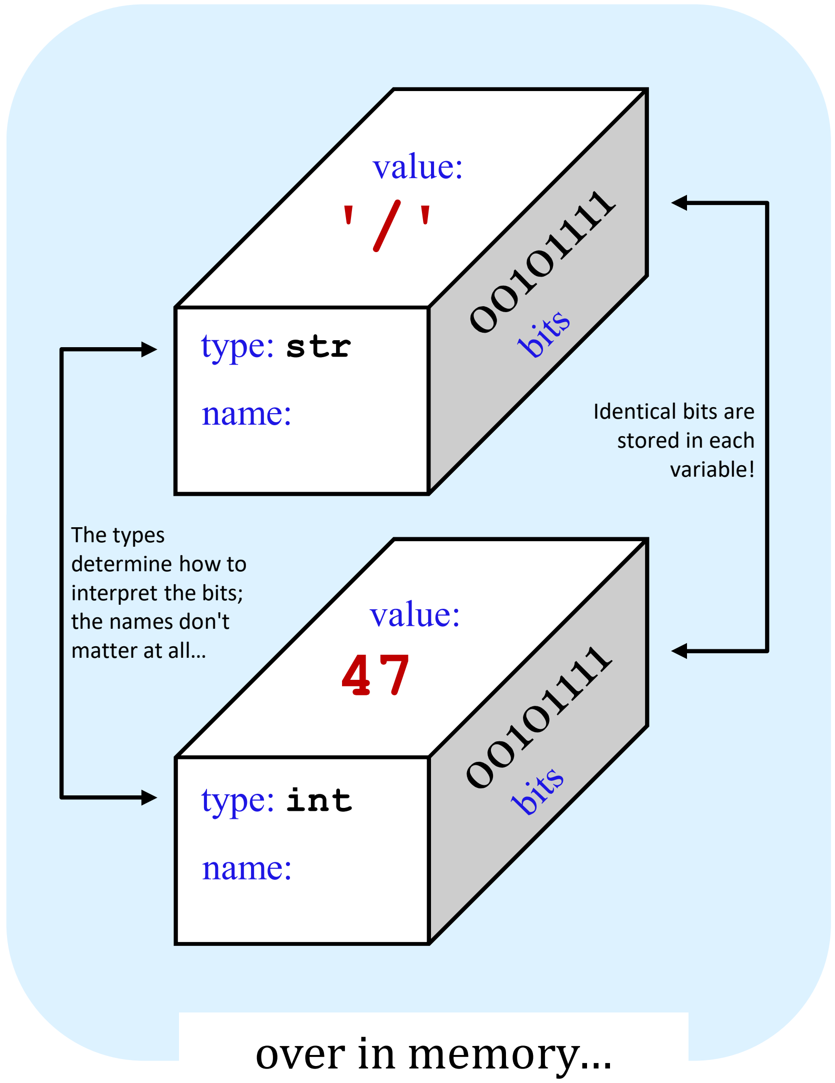
Binary	Dec	Hex	Glyph
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1

$\underbrace{\hspace{6em}}$

1 byte

8 bits

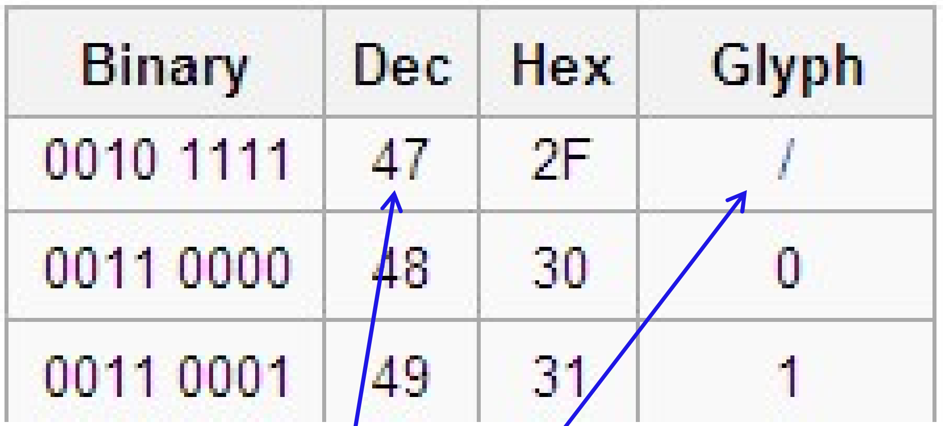
The SAME bits represent an integer or a string, depending on type: `int` or `str`


value:
'/'
00101111
bits
type: str
name:
Identical bits are stored in each variable!
The types determine how to interpret the bits; the names don't matter at all...
value:
47
00101111
bits
type: int
name:

over in memory...

Unicode + ASCII

In Python, **`chr`** and **`ord`** convert to/from Unicode + ASCII



Binary	Dec	Hex	Glyph
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1

$\underbrace{\text{0011 0001}}_{}$

1 byte

8 bits

The SAME bits represent an integer or a string, depending on type: `int` or `str`

convert # to char

`chr`

$\updownarrow$

`ord`

convert char to #

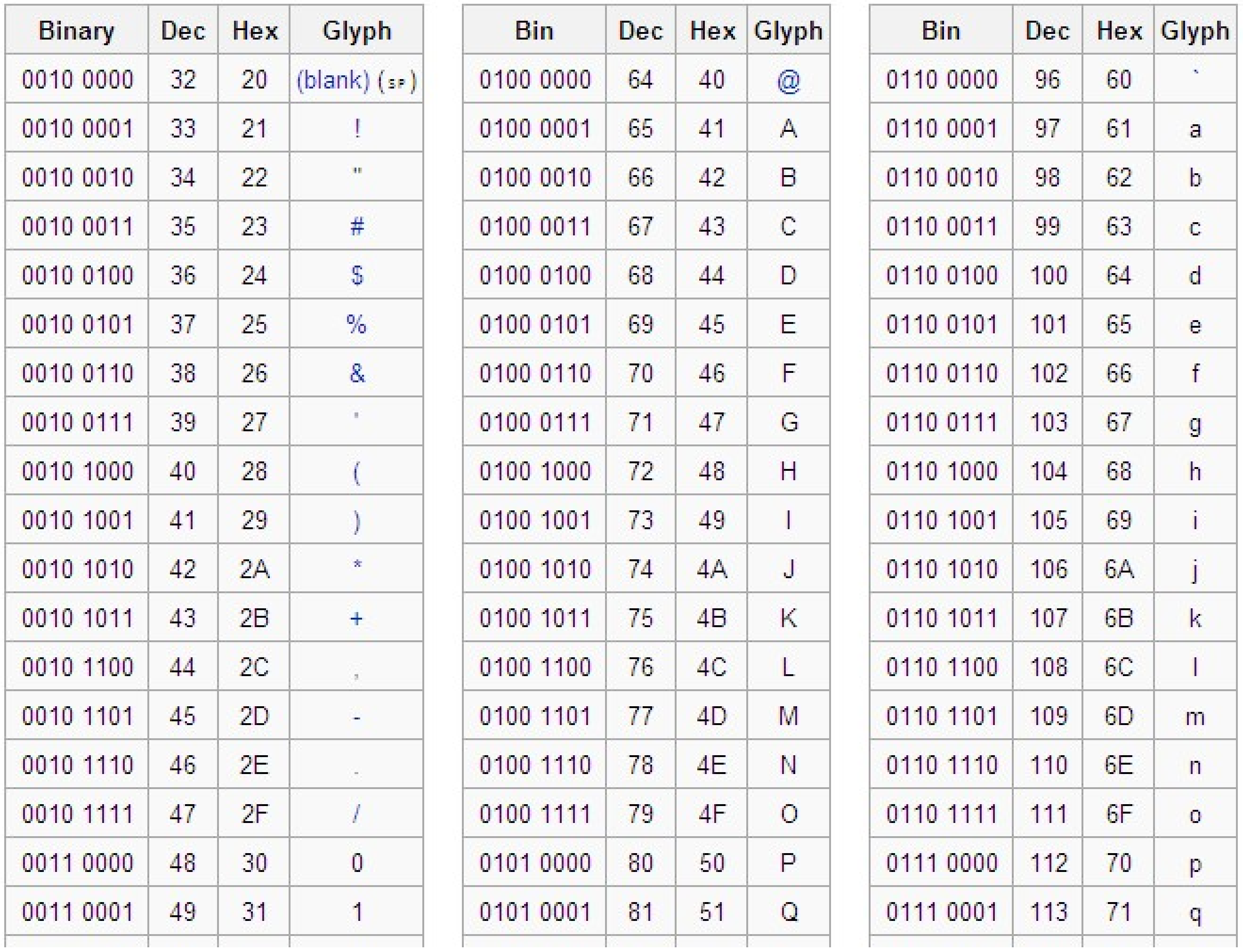
ASCII ⊂ Unicode

chr(8834)

convert # to char

$$\begin{tikzcd} \texttt{chr} \arrow[d, leftrightarrow] \\ \texttt{ord} \end{tikzcd}$$

convert char to #



Binary	Dec	Hex	Glyph
0010 0000	32	20	(blank) (SP)
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1

Bin	Dec	Hex	Glyph
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q

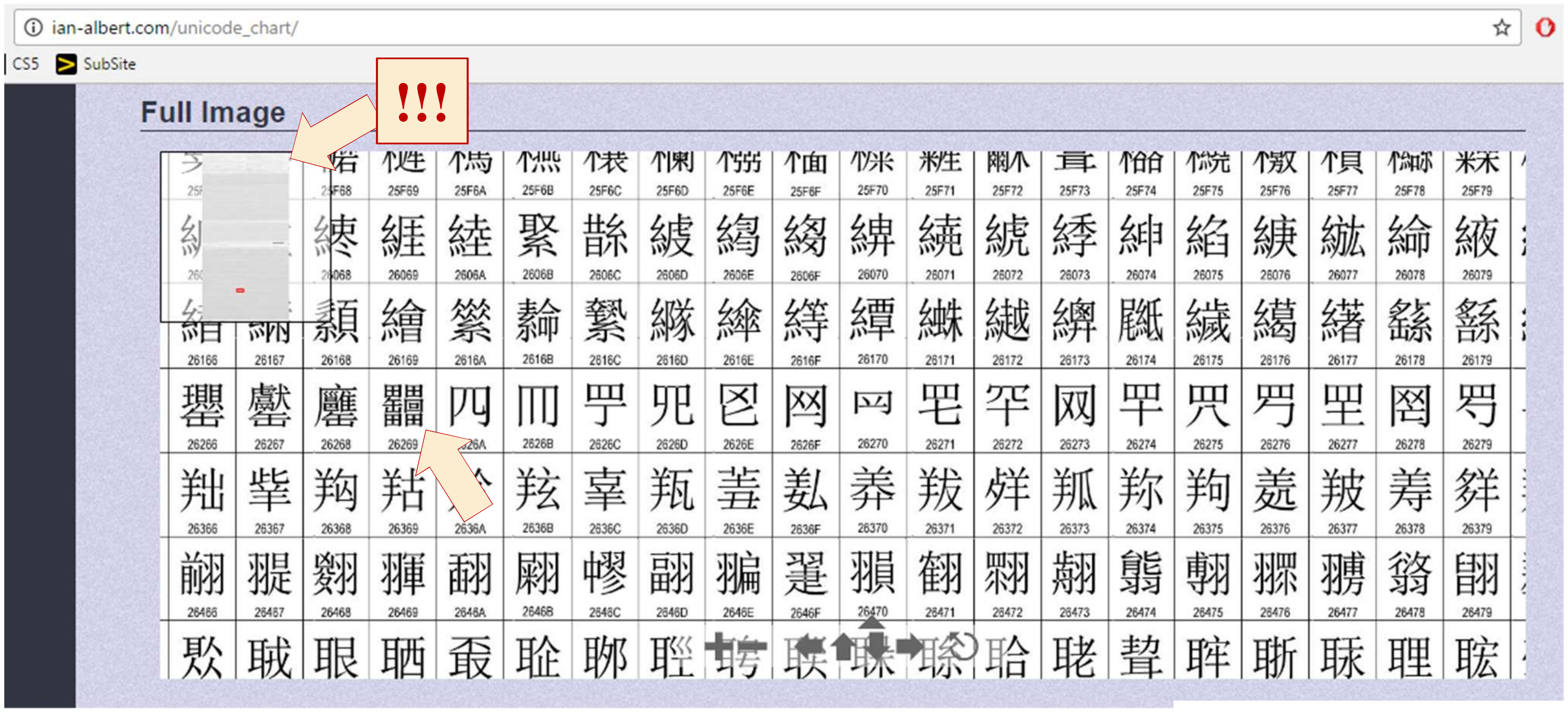
Bin	Dec	Hex	Glyph
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q

Julius spr'15

This is why `'CS'` `<` `'clear'` !

Unicode

Universal Character Encoding



http://ian-albert.com/unicode_chart/

Some fun characters...

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

My favorite is chr(1661)

on Win10: chcp 65001

Rot13

a useful and illustrative starting point...

```
abcdefghijklmnopqrstuvwxyz
97  99  101  103  105  107  109  111  113    117  119    122

ABCDEFGHI                  TUVWXYZ
65  67                      85  87    90
```

We'll build rot13 as a starting point...

```
                   put 'z'
```

adding 13

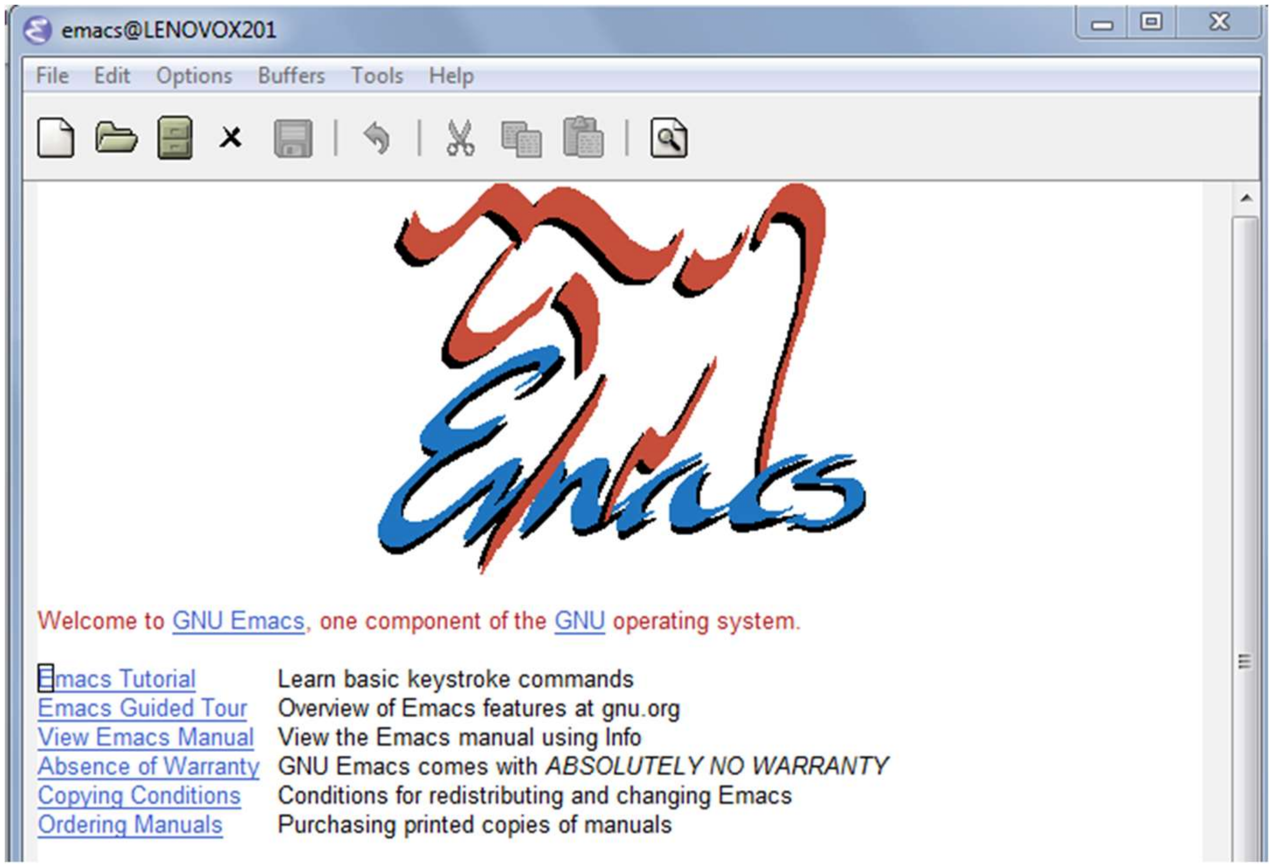
```
ot13('n') should output 'a'
rot13('W') should output 'J'
```

wrapping

```
rot13(' ') should output ' '
rot13('<') should output '<'
```

spaces + other characters

`rot13`'s surprising history...



Welcome to GNU Emacs, one component of the GNU operating system.

Emacs Tutorial	Learn basic keystroke commands
Emacs Guided Tour	Overview of Emacs features at gnu.org
View Emacs Manual	View the Emacs manual using Info
Absence of Warranty	GNU Emacs comes with *ABSOLUTELY NO WARRANTY*
Copying Conditions	Conditions for redistributing and changing Emacs
Ordering Manuals	Purchasing printed copies of manuals

rot13's key ideas...

convert # to char

$$\begin{tikzcd}\texttt{chr} \arrow[d, leftrightarrow] \\ \texttt{ord}\end{tikzcd}$$

convert char to #

What is `ord('U')//2`?

What is `chr( ord('i')+13 )`?

What is `chr( ord('W')+13 )`?

how do we wrap?

```
chr value  abcdefghijklmnopqrstuvwxyz
           97  99  101 103 105 107 109 111 113 115 117 119   122   ord value

chr value  ABCDEFGHIJKLMNOPQRSTUVWXYZ
           65  67  69  71  73  75  77  79  81  83  85  87    90    ord value
```

Writing Rot13

any single character, c

```
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':     # upper-case test!



    else:


```

(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?

Writing Rot13

any single character, c

```
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?

use n instead of 13

Extra: How would you rotate n places, instead of 13?

Caesar Cipher: *decipher*

Caesar

Brutus

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

S1

S2

```
>>> 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.'
```

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

PL

LAT

```
>>> decipher('gv vw dtwvg')
```

How!?

Decipher?

Strategies?

Algorithms?

Decipher?

Strategies?

Algorithms?

All possible decipherings

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

Decipher?

Strategies?

Algorithms?

All possible decipherings

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

Score them all

```
[0, 'gv vw dtwvg'],
[2, 'hw wx euxwh'],
[2, 'ix xy fvyxi'],
[0, 'jy yz gwzyj'],
[2, 'kz za hxazk'],
[4, 'la ab iybal'],
[0, 'mb bc jzcbm'],
[1, 'nc cd kadcn'],
[4, 'od de lbedo'],
[3, 'pe ef mcfep'],
[0, 'qf fg ndgfq'],
[2, 'rg gh oehgr'],
[2, 'sh hi pfihs'],
[3, 'ti ij qgjit'],
[2, 'uj jk rhkju'],
[1, 'vk kl     v'],
[0         w'],
             '],
          ony'],
     zo op wmpoz'],
[2, 'ap pq xnqpa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']
```

quantifying Englishness?

decPR(LAT)
decPR2(LAT)
decPR3(LAT)

Decipher?

Strategies?

Algorithms?

All possible decipherings

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

max!

Score them all

```
[0, 'gv vw dtwvg'],
[2, 'hw wx euxwh'],
[2, 'ix xy fvyxi'],
[0, 'jy yz gwzyj'],
[2, 'kz za hxazk'],
[4, 'la ab iybal'],
[0, 'mb bc jzcbm'],
[1, 'nc cd kadcn'],
[4, 'od de lbedo'],
[3, 'pe ef mcfep'],
[0, 'qf fg ndgfq'],
[2, 'rg gh oehgr'],
[2, 'sh hi pfihs'],
[3, 'ti ij qgjit'],
[2, 'uj jk rhkju'],
[1           silkv'],
              jmlw'],
              xnmx'],
               ony'],
             wmpoz'],
     'ap pq xnqpa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']
```

yields the "most English" phrase

```
decPR(LAT)
decPR2(LAT)
decPR3(LAT)
```

Measuring *Englishness*

Very English-y

higher scores

quantifying Englishness?

lower scores

Not 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"

All of these sound good to me!

Decipher?

Strategies?

Algorithms?

All possible decipherings

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

Score them all

max!

```
[0, 'gv vw dtwvg'],
[2, 'hw wx euxwh'],
[2, 'ix xy fvyxi'],
[0, 'jy yz gwzyj'],
[2, 'kz za hxazk'],
[4, 'la ab iybal'],
[0, 'mb bc jzcbm'],
[1, 'nc cd kadcn'],
[4, 'od de lbedo'],
[3, 'pe ef mcfep'],
[0, 'qf fg ndgfq'],
[2, 'rg gh oehgr'],
[2, 'sh hi pfihs'],
[3, 'ti ij qgjit'],
[2, 'uj jk rhkju'],
[1            ilkv'],
               mlw'],
               nmx'],
               ony'],
             wmpoz'],
    'ap pq xnqpa'],
[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

decPR(LAT)
decPR2(LAT)
decPR3(LAT)

Decipher?

Strategies?

Algorithms?

decPR(LAT)
decPR2(LAT)
decPR3(LAT)

All possible decipherings

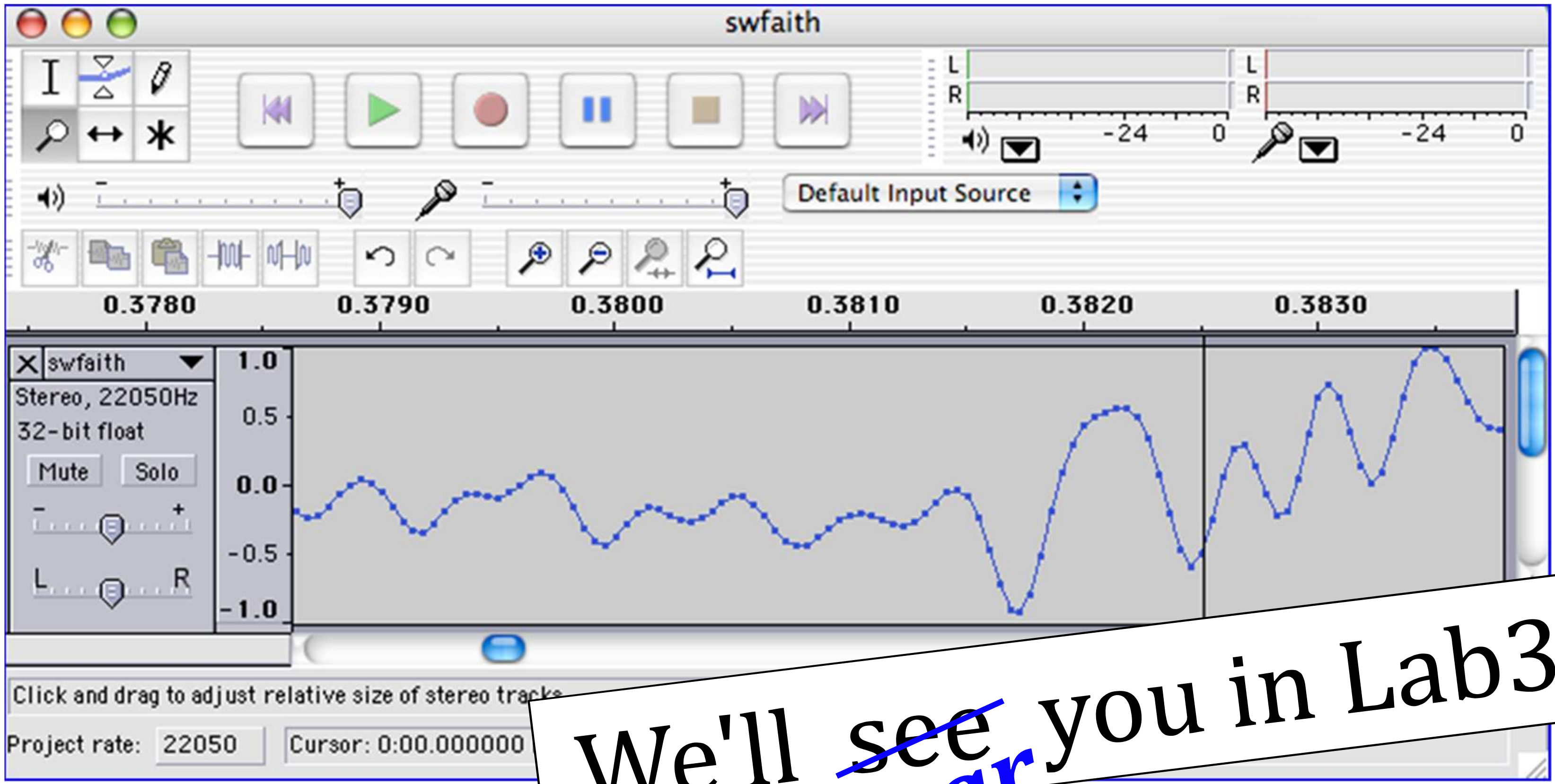
```
gv vw dtwvg
hw wx euxwh
ix xy fvyxi
jy yz gwzyj
kz za hxazk
la ab iybal
mb bc jzcbm
nc cd kadcn
od de lbedo
pe ef mcfep
qf fg ndgfq
rg gh oehgr
sh hi pfihs
ti ij qgjit
uj jk rhkju
vk kl silkv
wl lm tjmlw
xm mn uknmx
yn no vlony
zo op wmpoz
ap pq xnqpa
bq qr yorqb
cr rs zpsrc
ds st aqtsd
et tu brute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

Scores

```
[6.9e-05, 'gv vw dtwvg'],
[3.6e-05, 'hw wx euxwh'],
[1.4e-07, 'ix xy fvyxi'],
[8.8e-11, 'jy yz gwzyj'],
[7.2e-10, 'kz za hxazk'],
[0.01503, 'la ab iybal'],
[3.7e-08, 'mb bc jzcbm'],
[0.00524, 'nc cd kadcn'],
[0.29041, 'od de lbedo'],
[0.00874, 'pe ef mcfep'],
[7.3e-07, 'qf fg ndgfq'],
[0.06410, 'rg gh oehgr'],
[0.11955, 'sh hi pfihs'],
[3.1e-06, 'ti ij qgjit'],
[1.1e-08, 'uj jk rhkju'],
[2.6e-0[illegible] silkv'],
[illegible] tjmlw'],
[illegible] uknmx'],
[illegible] lony'],
[illegible] op wmpoz'],
[illegible]-07, 'ap pq xnqpa'],
[5.7e-08, 'bq qr yorqb'],
[0.00024, 'cr rs zpsrc'],
[0.02060, 'ds st aqtsd'],
[0.45555, 'et tu brute'],
[0.00011, 'fu uv csvuf']
```

max!

"Englishness" based on letter-probabilities



Click and drag to adjust relative size of stereo tracks

Project rate: 22050

Cursor: 0:00.000000

We'll ~~see~~ *hear* you in Lab3 !

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