BR 5 Snczx

Gesundheit!

Algorithms

Englishness...
Classifying life
Removing/Sorting
and *Jotto!*

HW 3

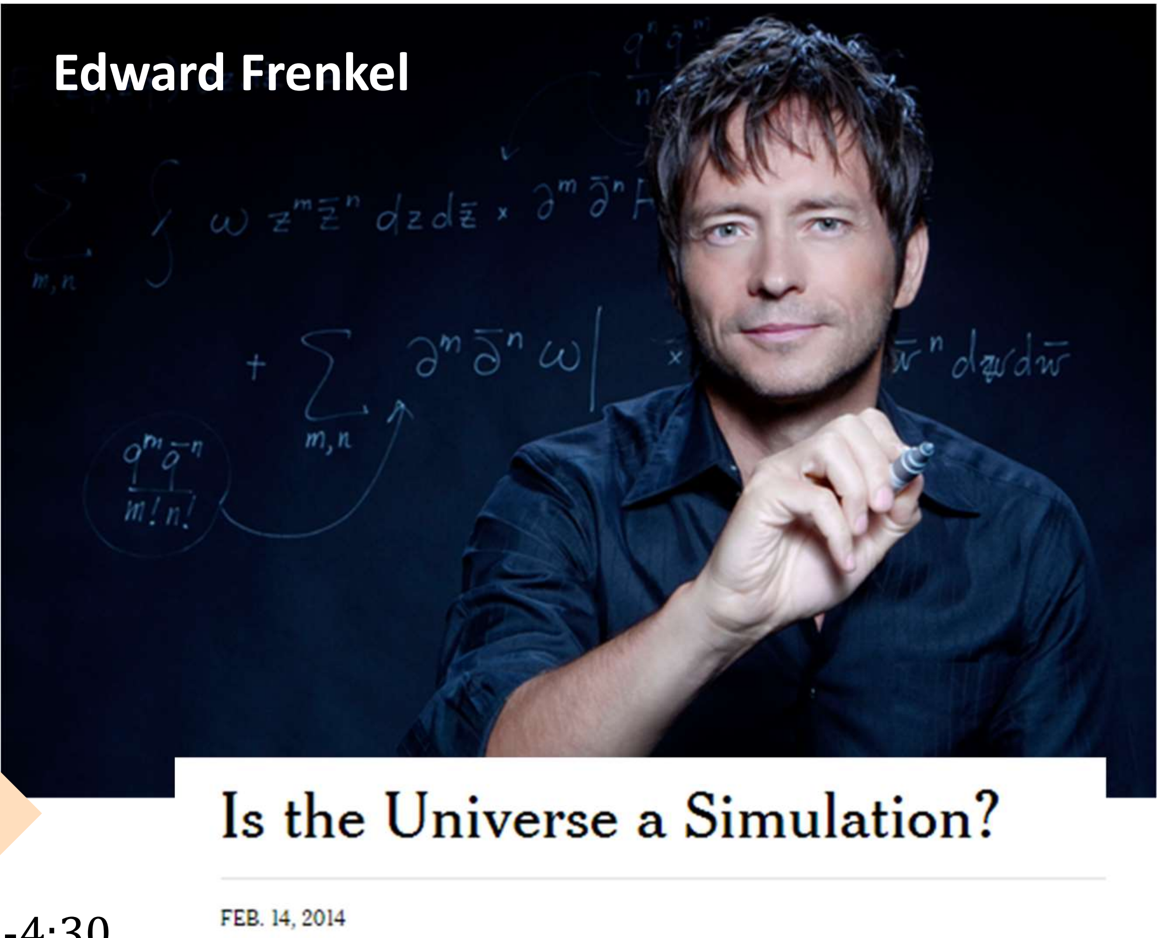
Hw #3 due **Monday,** 11:59

Sound Lab!

Several algorithms ...

Office Hrs.!

Friday, 2:30-4:30,
HMC's LAC lab...

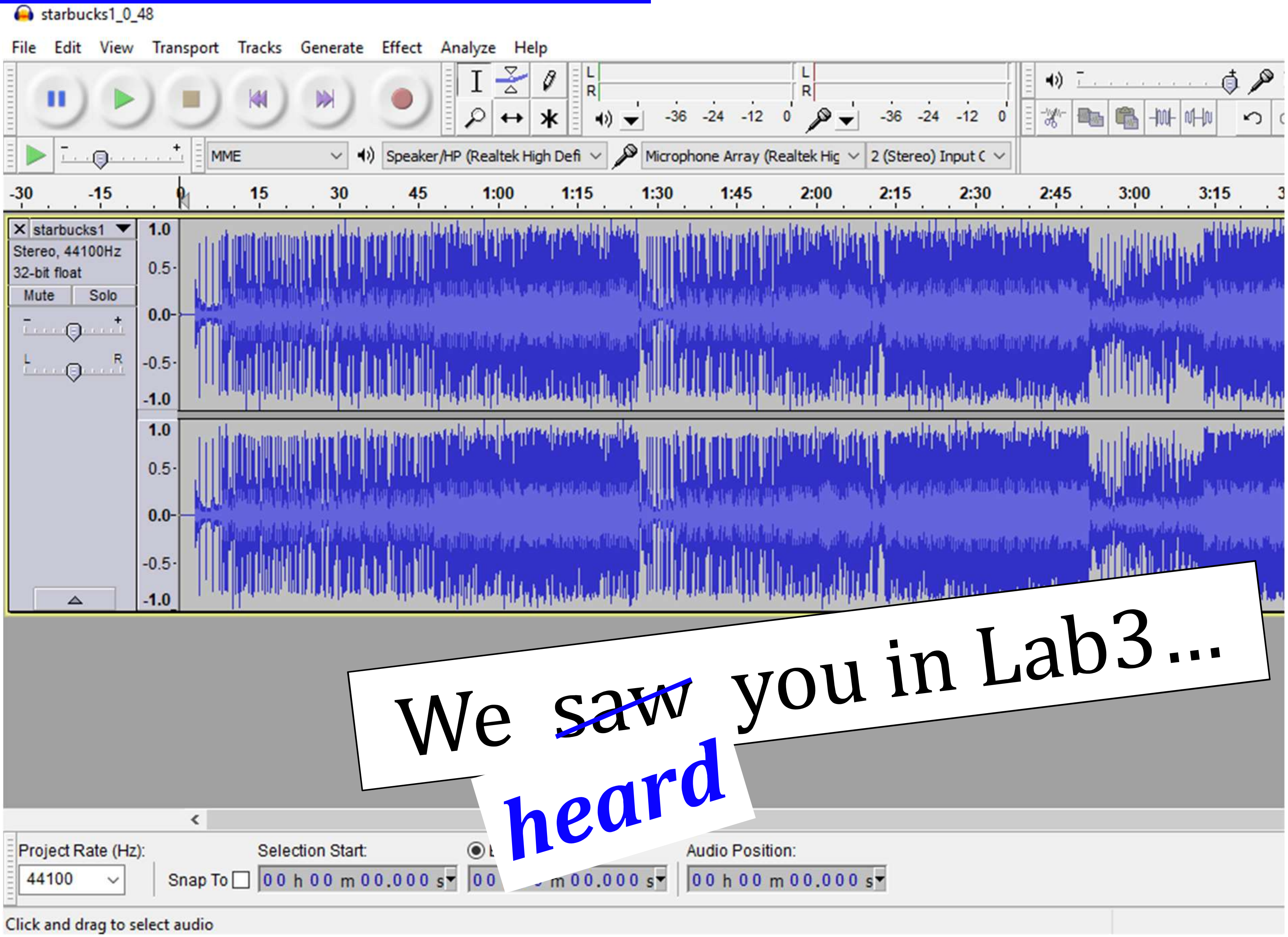


Edward Frenkel

Is the Universe a Simulation?

FEB. 14, 2014

Sound + Starbucks!



starbucks1_0_48

Speaker/HP (Realtek High Defi

Microphone Array (Realtek Hig

2 (Stereo) Input C

starbucks1

Stereo, 44100Hz

32-bit float

Project Rate (Hz): 44100

Snap To □

Selection Start: 00 h 00 m 00.000 s

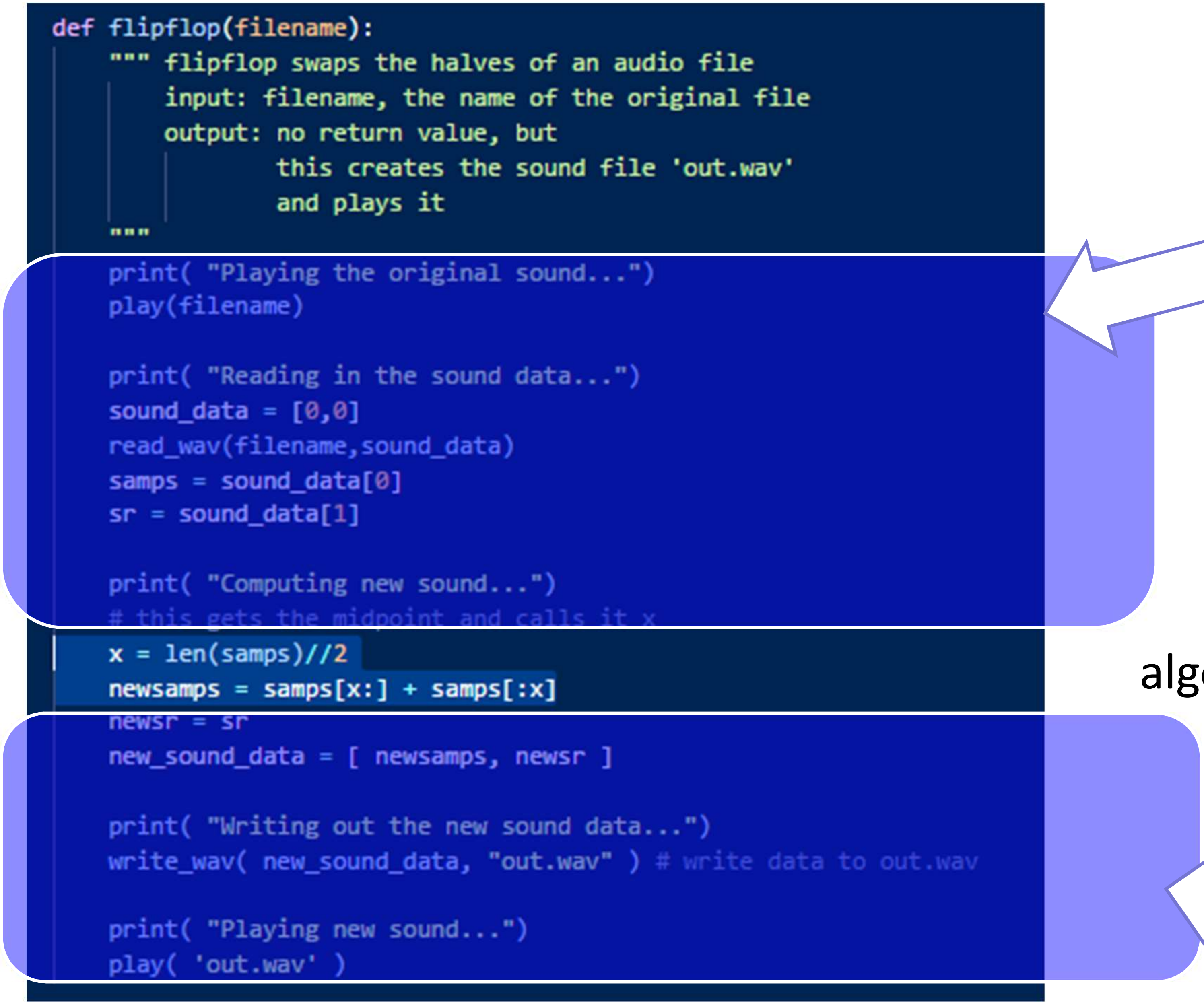
● 00 h 00 m 00.000 s

Audio Position: 00 h 00 m 00.000 s

Click and drag to select audio

We ~~saw~~ **heard** you in Lab3...

Take-away ~ Lab3



```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 – important, but less algorithmic

algorithmic stuff

"outro" stuff

BR 5 Snczx

Gesundheit!

Algorithms

Englishness...
Classifying life
Removing/Sorting
and *Jotto!*

HW 3

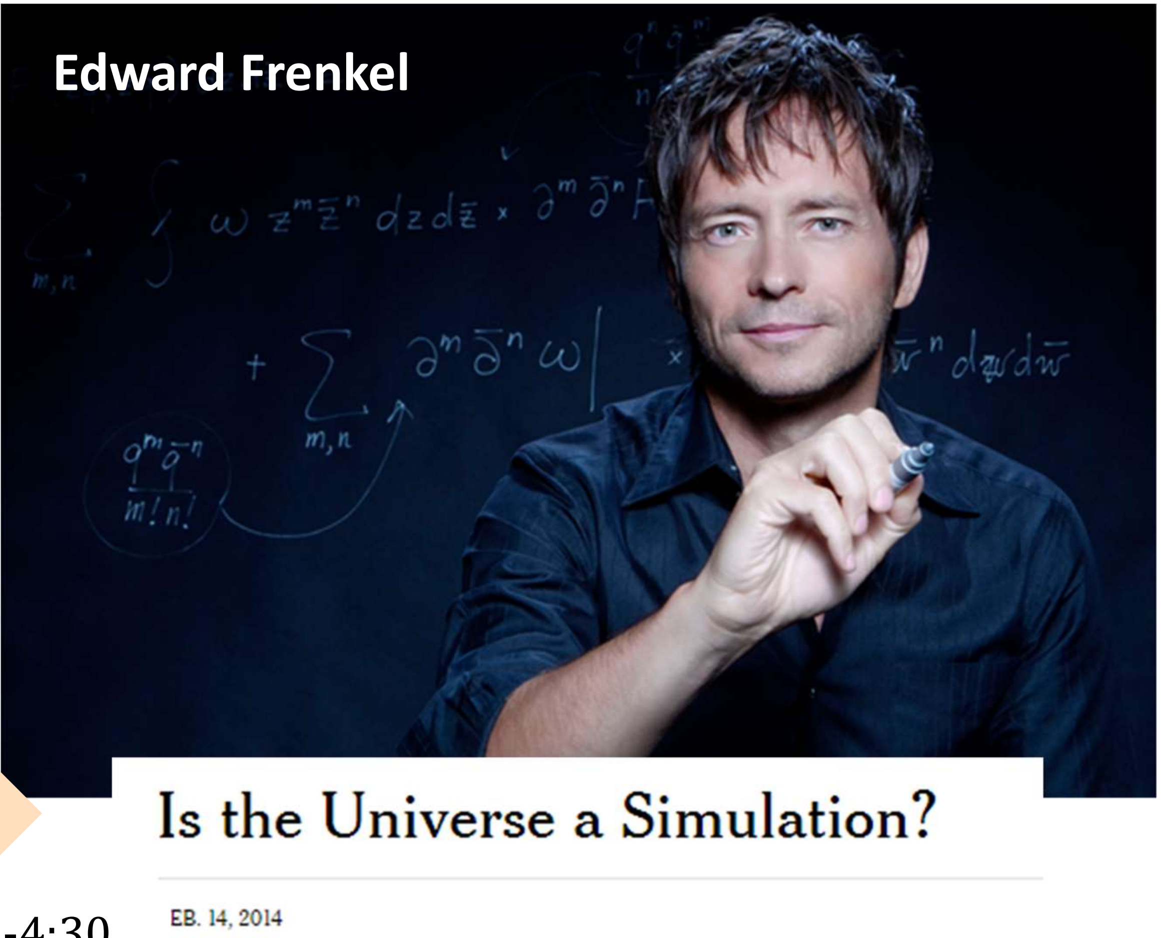
Hw #3 due **Monday,** 11:59

Sound Lab!

Several algorithms ...

Office Hrs.!

Friday, 2:30-4:30,
HMC's LAC lab...



Edward Frenkel

Is the Universe a Simulation?

EB. 14, 2014

BR 5 Snczx

Gesundheit!

Algorithms

Englishness...
Classifying life
Removing/Sorting
and *Jotto!*

HW 3

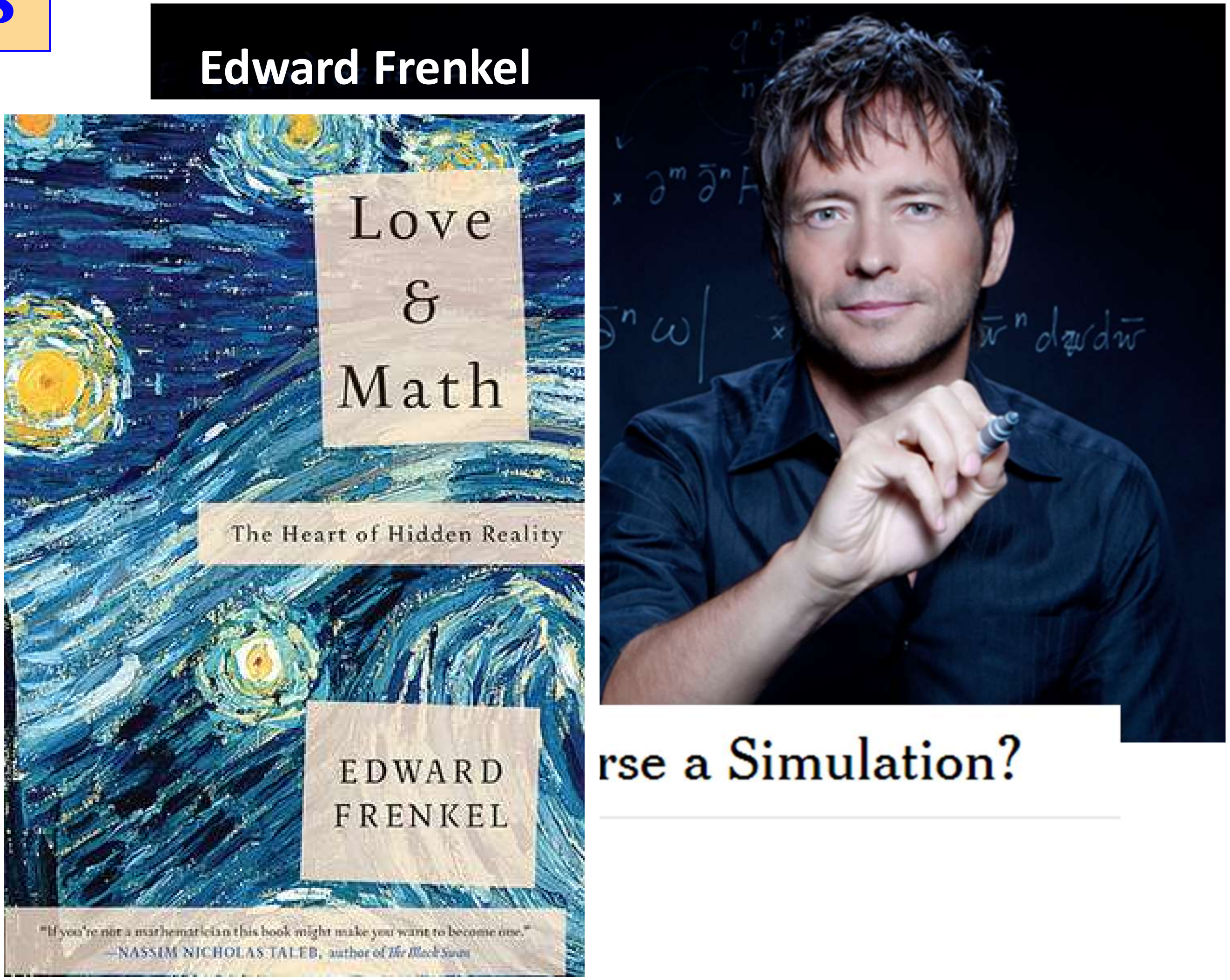
Hw #3 due **Monday, 1**

Sound Lab!

Several <u>algorithms</u> .

Office Hrs.!

Fri
HM


Edward Frenkel
Love & Math
The Heart of Hidden Reality
EDWARD FRENKEL
"If you're not a mathematician this book might make you want to become one."
—NASSIM NICHOLAS TALEB, author of The Black Swan

rse a Simulation?

Soundbites lab discovery!

Inbox x

gma

9:56 PM (14 hours ago)

Hi Professor Dodds,

While

Clic

Proj

missy.wav

Today in CS5:

chr(9829)

The ♥ of CS (and CSers...)

Algorithms!

chcp 65001

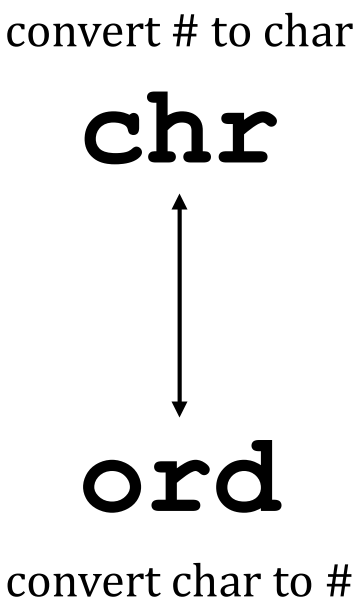
Caesar Cipher: `encipher`

`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'
```

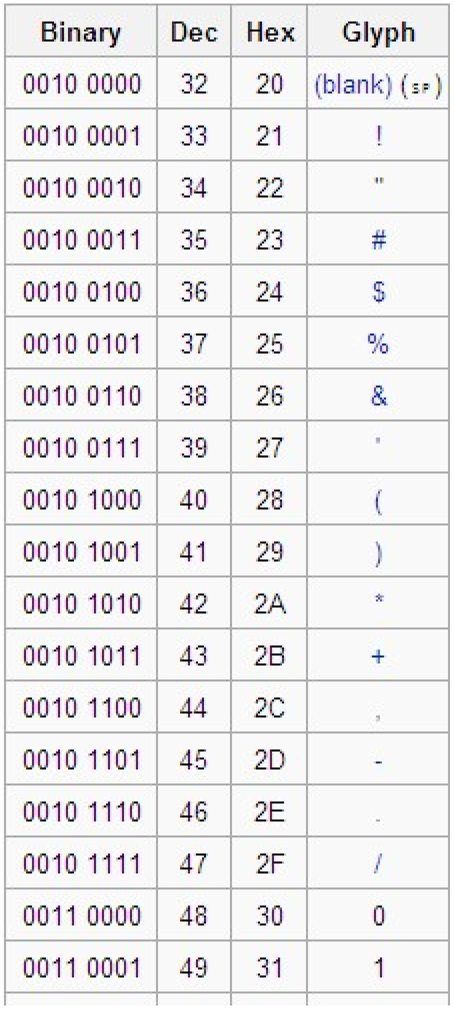
ASCII *and* Unicode


convert # to char

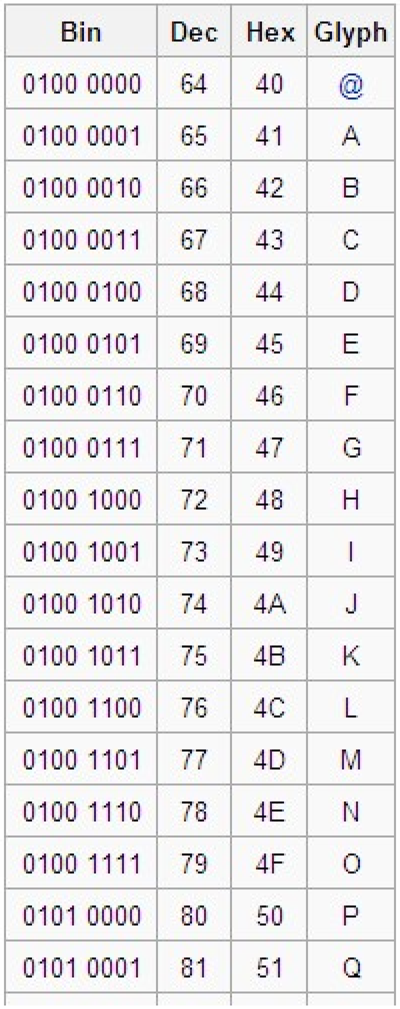
chr

ord

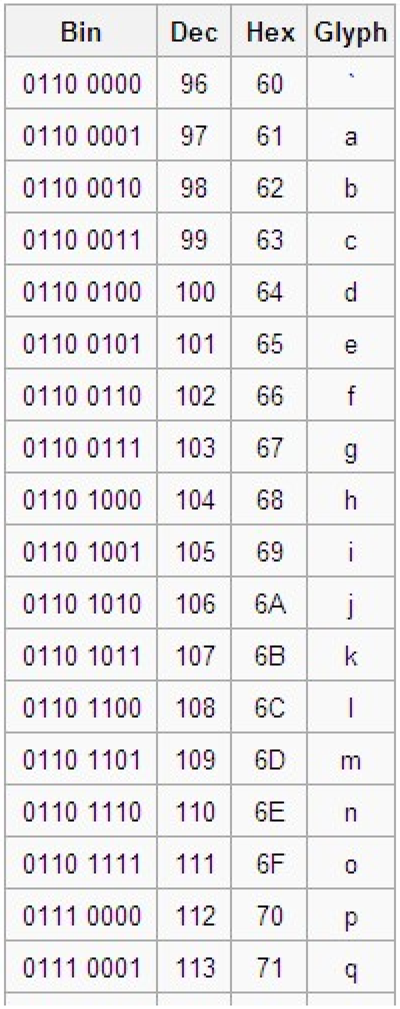
convert char to #



Binary	Dec	Hex	Glyph
0010 0000	32	20	(blank) (␠)
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'` !

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 for '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

Caesar Cipher: **encipher**

Brutus

```
>>> encipher('Bzdrzq bhogdq? H oqdedq Bzdrzq rzkzc.',25)          S1
'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
'Caesar cipher? I pre

>>> encipher('Hu lkbjh          pz doha ylthpuz hmaly dl mvynla '\
             'lclyfaopun dl ohcl slhyulk.',19)
'An education is what remains after we forget everything we
have learned.'
                                                                   S2
```

model for this problem:

transcribe from hw#1

Caesar

Caesar Cipher: decipher

Brutus

S1

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

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, PL2

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

LAT

But ***how*** ?

Decipher?

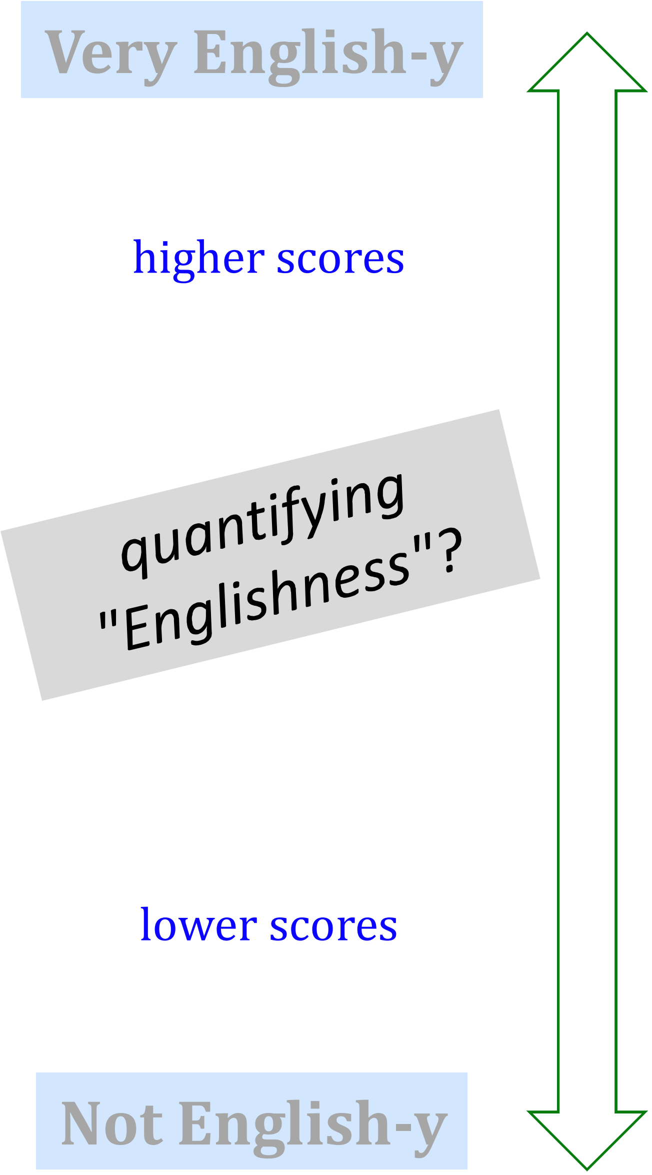
All possible decipherings

```
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
```

Strategies?

Algorithms?

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 t
xm mn u
yn no v
zo op w
ap pq x
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, 'vk kl silkv'],
[0, 'wl lm tjmlw'],
[1, 'xm mn uknmx'],
[2, 'yn no vlony'],
[3, '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']
```

"Englishness" score ~ the #-of-vowels

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
qr yorqb
cr rs zpsrc
ds st aqtsd
et tu brute
fu uv csvuf
```

"Englishness" based on letter-probabilities

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-05, 'vk kl silkv'],
[0.00012, 'wl lm tjmlw'],
[3.1e-06, 'xm mn uknmx'],
[0.02011, 'yn no vlony'],
[1.5e-06, 'zo op wmpoz'],
[1.9e-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!

Decipher?

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 mcf
      rhkju
vk kl silkv
wl lm tjml
xm
a
b       qb
cr rs zpsrc
ds st aqtsd
et tu brute
fu uv csvuf
```

S
c

```
[27, 'gv vw dtwvg']
[38, 'hw wx euxwh']
[42, 'ix xy fvyxi']
[54, 'jy yz gwzyj']
[54, 'kz za hxazk']
[16, 'la ab iybal']
[39, 'mb bc jzcbm']
[21, 'nc cd kadcn']
      'od de lbedo']
      'pe ef mcfep']
      'qf fg ndgfq']
      'rg gh oehgr']
      'sh hi pfihs']
[33, 'ti ij qgjit']
[41, 'uj jk rhkju']
[27, 'vk kl silkv']
[26, 'wl lm tjmlw']
[33, 'xm mn uknmx']
[18, 'yn no vlony']
[36, 'zo op wmpoz']
[40, 'ap pq xnqpa']
[43, 'bq qr yorqb']
[24, 'cr rs zpsrc']
[20, 'ds st aqtsd']
[11, 'et tu brute']
[23, 'fu uv csvuf']
```

min!

Strategies?

*Using the **LoL** technique to score each rotation's "Englishness"*

Algorithms?

"Englishness" based on scrabble-scoring!

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

Design…

design of ***what****?*

The ♥ of CS

(and CSers…)

Algorithms!

Design…

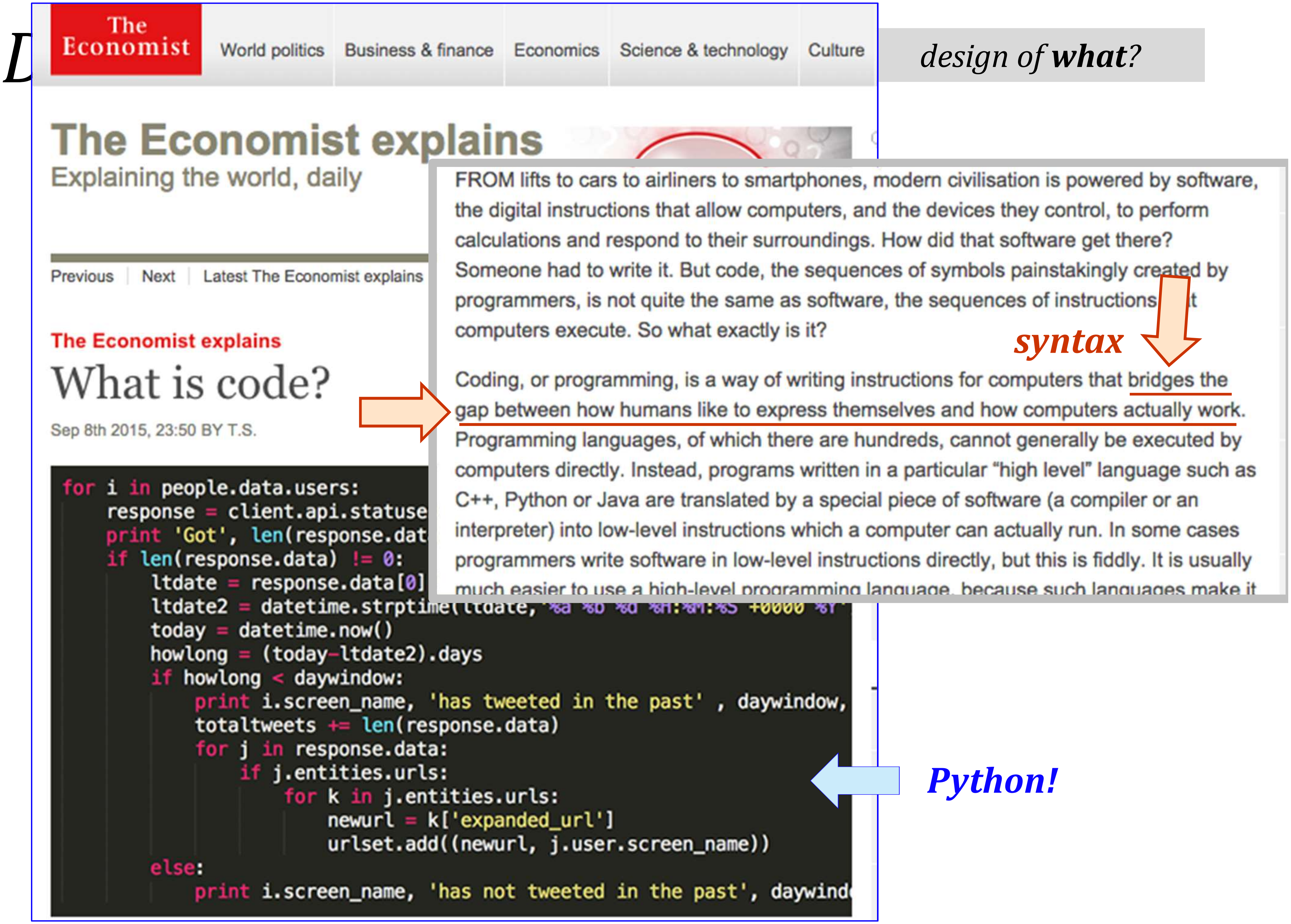
*design of **what**?*

~~*Code?*~~

syntax

D

design of ***what***?



The Economist explains

Explaining the world, daily

The Economist explains

What is code?

Sep 8th 2015, 23:50 BY T.S.

```
for i in people.data.users:
    response = client.api.statuse
    print 'Got', len(response.dat
    if len(response.data) != 0:
        ltdate = response.data[0]
        ltdate2 = datetime.strptime(ltdate, '%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

FROM lifts to cars to airliners to smartphones, modern civilisation is powered by software, the digital instructions that allow computers, and the devices they control, to perform calculations and respond to their surroundings. How did that software get there? Someone had to write it. But code, the sequences of symbols painstakingly created by programmers, is not quite the same as software, the sequences of instructions that computers execute. So what exactly is it?

Coding, or programming, is a way of writing instructions for computers that bridges the gap between how humans like to express themselves and how computers actually work. Programming languages, of which there are hundreds, cannot generally be executed by computers directly. Instead, programs written in a particular "high level" language such as C++, Python or Java are translated by a special piece of software (a compiler or an interpreter) into low-level instructions which a computer can actually run. In some cases programmers write software in low-level instructions directly, but this is fiddly. It is usually much easier to use a high-level programming language, because such languages make it

syntax

Python!

Design…

design of ***what****?*

~~*Code?*~~

syntax

Algorithms!

ideas!

Algorithm Design...

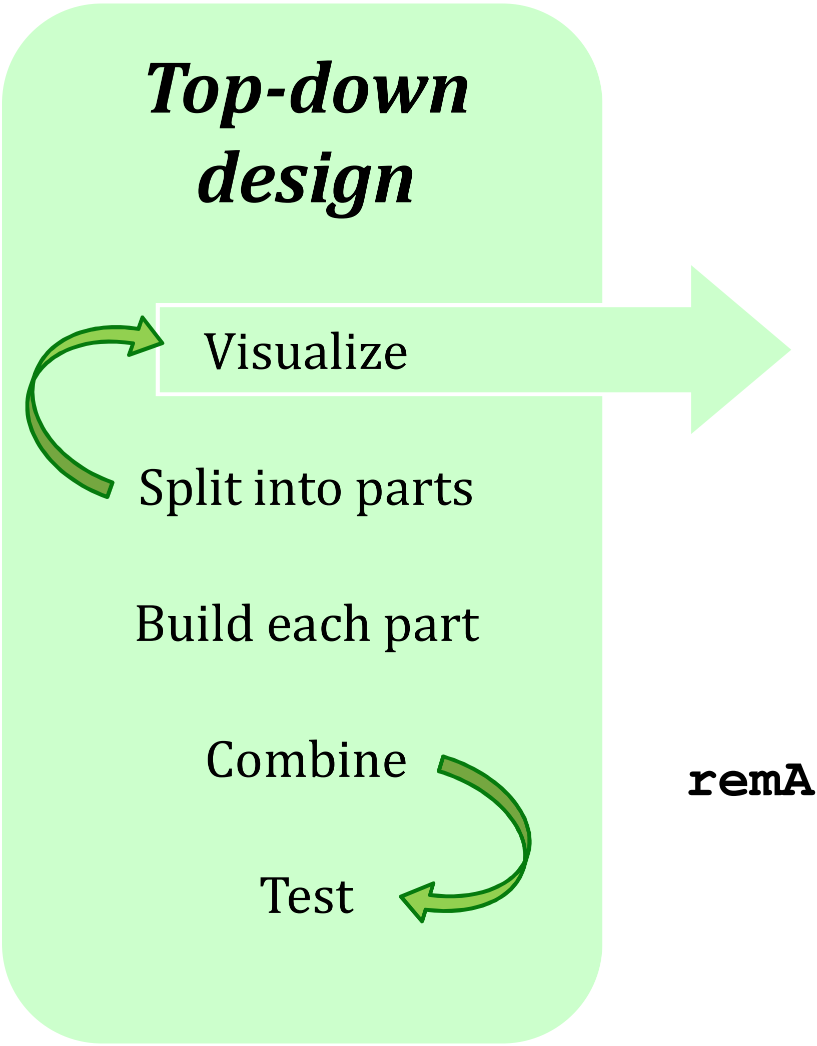
`remAll(e,L)`

remove all e's from L

Design...

```
remAll(e,L)
```

remove all e's from L



Top-down design

Visualize

Split into parts

Build each part

Combine

Test

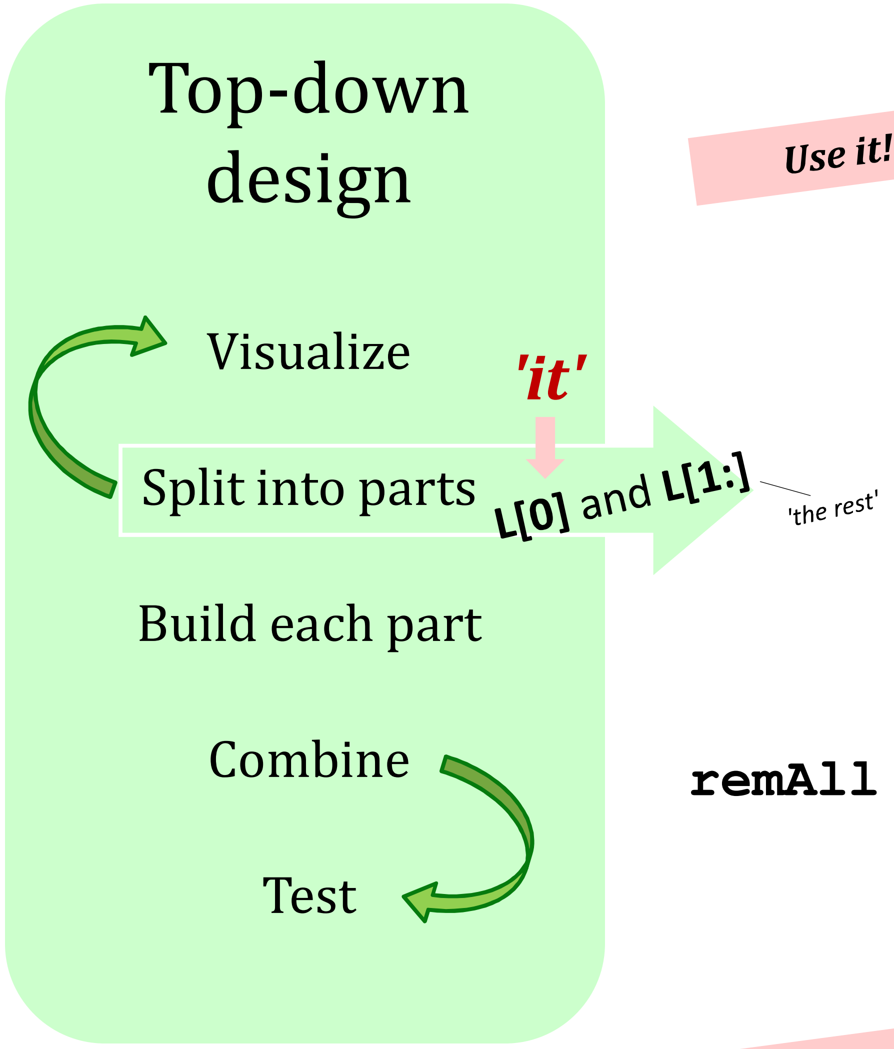
```
remAll(42,[5,7,42,8,42])
```

```
[5,7,8]
```

```
remAll('q','qaqqlqqiqqiiqeqqnqs')
```

```
'aliiiens'
```

Design...



```
remAll(e,L)
```

remove all e's from L

Use it!

it

'the rest'

```
remAll(42,[5,7,42,8,42])
```

```
[5,7,8]
```

it

'the rest'

```
remAll('q','qaqqlqqiqqiiqeqqnqs')
```

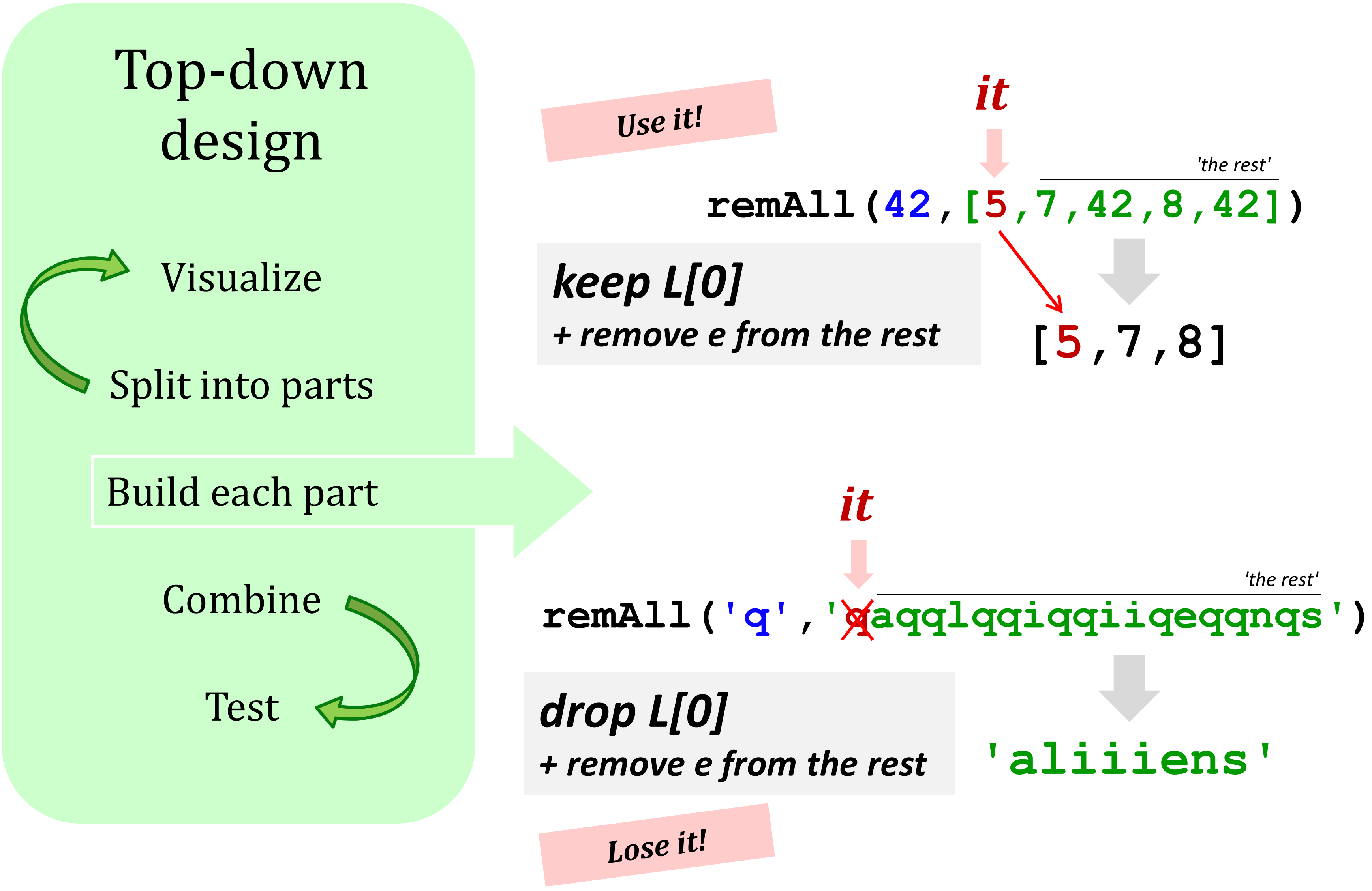
```
'aliiiens'
```

Lose it!

Design...

`remAll(e,L)`

remove all e's from L



Top-down design

Visualize

Split into parts

Build each part

Combine

Test

Use it!

it

'the rest'

`remAll(42,[5,7,42,8,42])`

keep L[0]

+ remove e from the rest

`[5,7,8]`

it

'the rest'

remAll('q','~~q~~aqqlqqiqqiiqeqqnqs')

drop L[0]

+ remove e from the rest

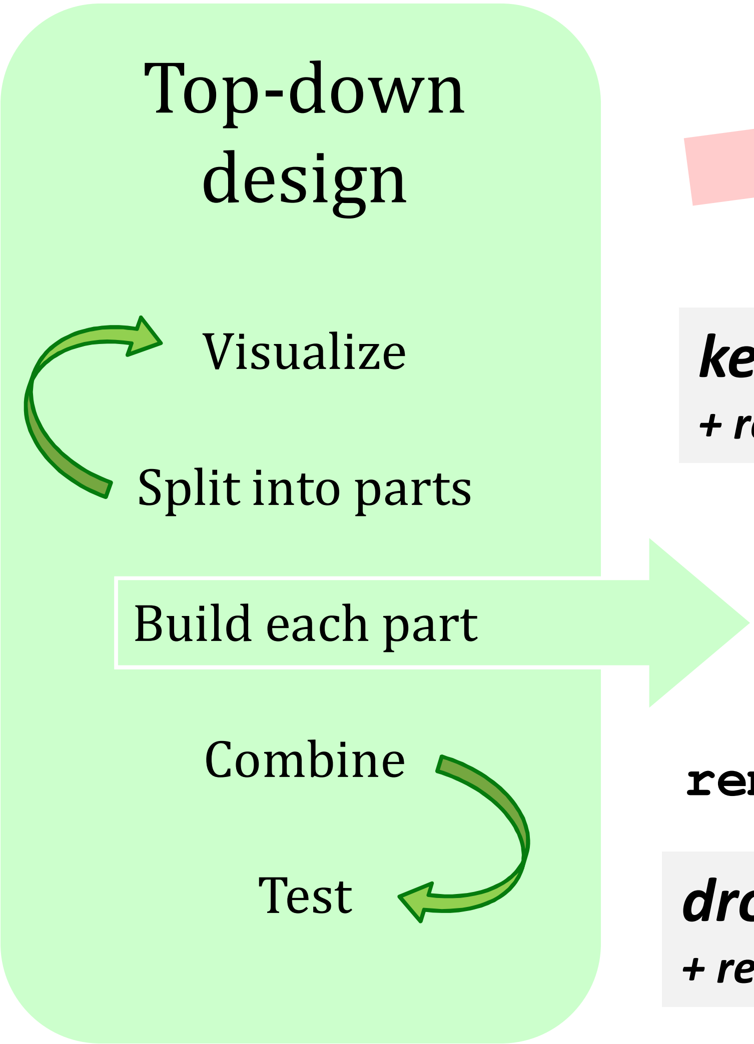
`'aliiiens'`

Lose it!

Design…

`remAll(e,L)`

remove all e's from L



Top-down design

Visualize

Split into parts

Build each part

Combine

Test

Use it!

it

`remAll(42`

keep L[0]

+ remove e fr

Use it!

- or -

it

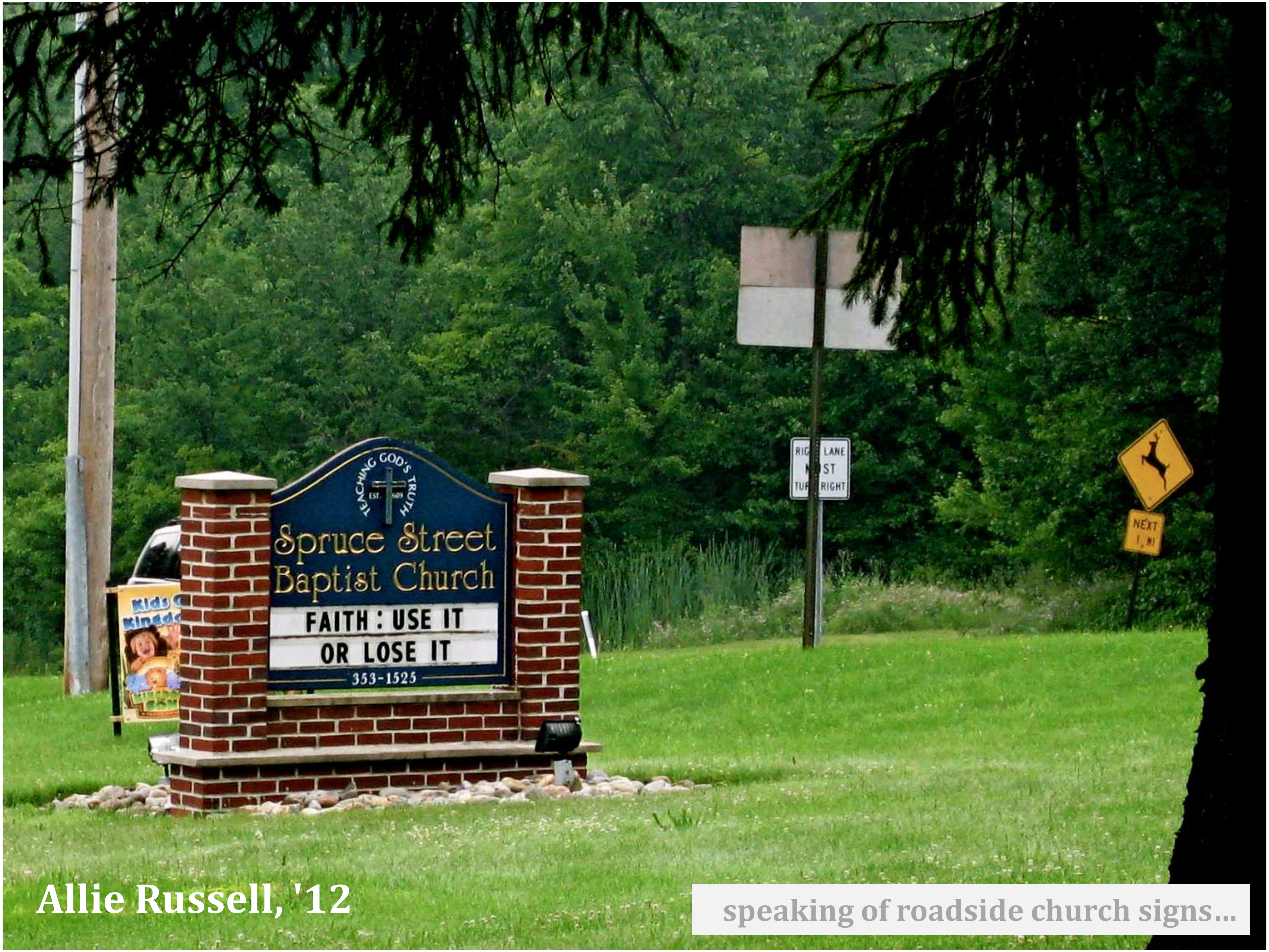
`remAll('q','~~q~~agq`

drop L[0]

+ remove e fro

Lose it.

Lose it!



Allie Russell, '12

speaking of roadside church signs…

Career Fair '16

Allie Russell, '12

Design ~ *code*

`remAll(e,L)`

remove all e's from L

Top-down design

Re-Visualize *in syntax*!?

If there are no elements or characters in L, we're done – return L itself!

```
def remAll( e, L ):
    """ removes all        from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return          remAll(e,L[1:])
```

Design ~ *code*

`remAll(e,L)`

remove all e's from L

Top-down design

[7,5,42]

Re-Visualize *in syntax*!?

```
def remAll( e, L ):
    """ removes all            L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return                 remAll(e,L[1:])
```

If **it** is not e,

USE it (keep it in the return value)

AND remove all of the e's from the rest of L!

Design ~ *code*

`remAll(e,L)`

remove all e's from L

Top-down design

[7,5,42]

Re-Visualize *in syntax*!?

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:       +               e,L
    else:
        return                    remAll(e,L[1:])
```

If **it** *is* e,

LOSE it (don't keep it in the return value)

AND still remove all of the e's from the rest of L!

Design ~ *code*

`remAll(e,L)`

That's it. *Algorithmic expression* ~ it's what CSers (think they) do.

Re-Visualize *in syntax*!?

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return          remAll(e,L[1:])
```

Name(s):

Quiz

cs5 hrs last week

Algorithm design

Don't start yet...

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

1

Change **remAll** so that it removes only ***one*** **e** from **L**. (We could call it **remOne**.)

`remOne(8,[7,8,9,8])` ➜ `[7,9,8]`

2

Make *more* changes to **remAll** so that it removes all of the elements ***up to and including the first*** **e** in **L**. (We could call it **remUpto**.)

`remUpto('d','coded')` ➜ `'ed'`

If e is not in L, remUpto should remove everything...

Hint: In both cases, what's needed is *mostly [illegible] stuff?*

```
def subseq( s, sbig ):
   """ returns True if s is a subseq. of sbig,
       False otherwise. Both are strings.
   """
   if s == '':
       return True
   elif
```

Challenge...

3 Write the other cases needed for **subseq**...

```
subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
```

Name(s):

Quiz

cs5 hrs last week

Algorithm design

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

1

Change **remAll** so that it removes only ***one*** **e** from **L**. (We could call it **remOne**.)

remOne(8,[7,8,9,8]) ➡ **[7,9,8]**

2

Make *more* changes to **remAll** so that it removes all of the elements ***up to and including the first*** **e** in **L**. (We could call it **remUpto**.)

remUpto('d','coded') ➡ **'ed'**

If e is not in L, remUpto should remove everything...

Hint: In both cases, what's needed is *mostly crossing stuff out*! *What stuff?*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig,
        False otherwise. Both are strings.
    """
    if s == '':
        return True
    elif
```

Challenge...

3 Write the other cases needed for **subseq...**

```
subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
```

Name(s):

Quiz

cs5 hrs last week

Algorithm design

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

1

Change **`remAll`** so that it removes only ***one*** **e** from **L**. (We could call it **`remOne`**.)

`remOne(8,[7,8,9,8])` ➡ **`[7,9,8]`**

2

Make *more* changes to **`remAll`** so that it removes all of the elements ***up to and including the first*** **e** in **L**. (We could call it **`remUpto`**.)

`remUpto('d','coded')` ➡ **`'ed'`**

If e is not in L, remUpto should remove everything...

Hint: In both cases, what's needed is *mostly crossing stuff out*! *What stuff?*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig,
        False otherwise. Both are strings.
    """
    if s == '':
        return True
    elif
```

Challenge...

3

Write the other cases needed for **subseq**...

```
subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
```

remAll insight

syntax

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

```
remAll(8, [7,8,9,8]) → [7,9]
           0 1 2 3
```

sharpening our model for where + how actions happen...

other **`rem`** examples...

`remAll(8, [7,8,9,8])`	→	`[7,9]`	**`remAll`**
`remAll('d', 'coded')`	→	`'coe'`	**`remAll`**
`remOne(8, [7,8,9,8])`	→	`[7,9,8]`	**`remOne`**
`remOne('d', 'coded')`	→	`'coed'`	**`remOne`**
`remUpto(8,[7,8,9,8])`	→	`[9,8]`	**`remUpto`**
`remUpto('d','coded')`	→	`'ed'`	**`remUpto`**

Subsequences

in order, but not necessarily ***adjacent***...

```
def subseq( s, sbig )
```

→ True or False?

s is the subsequence to find (or not)

sbig is the bigger string in which we are looking for **s**

T or **F**?

`subseq('', 'cataga')` → **True**

`subseq('ctg', 'cataga')` → **True**

`subseq('ctg', 'tacggta')` →

`subseq('aliens', 'always frighten dragons')` →

`subseq('trogdor', 'that dragon is gone for good')` →

Here there be NO dragons!

Why Are these True? or False?

Name(s):

Quiz

cs5 hrs last week

Algorithm design

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + re
    else:
        return remAll(e,L[1:])
```

1

Change **remAll** so that it removes only ***one*** **e** from **L**. (We could call it **remOne**.)

remOne(8,[7,8,9,8]) ➡ **[7,9,8]**

Hint: remove one thing for **remOne!**

2

Make *more* changes to **remAll** so that it removes all of the elements ***up to and including the first*** **e** in **L**. (We could call it **remUpto**.)

Hint: remove one ***more*** thing for **remUpto!**

remUpto('d','coded') ➡ **'ed'**

If e is not in L, remUpto should remove everything...

Hint: In both cases, what's needed is *mostly crossing stuff out! What stuff?*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig,
        False otherwise. Both are strings.
    """
    if s == '':
        return True
    elif
```

Challenge...

3 Write the other cases needed for **subseq...**

```
subseq('alg','magical')
 lse

subseq('alg','twasbrillig')
True
```

Hint: you'll need 3-4 cases total for **subseq.**

Name(s):

Quiz

cs5 hrs last week

Algorithm design

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

Try it out!

Hint: In both cases, what's needed is *mostly crossing stuff out! What stuff?*

1

Change **remAll** so that it removes only ***one*** **e** from **L**. (We could call it **remOne**.)

`remOne(8,[7,8,9,8])` ➡ `[7,9,8]`

2

Make *more* changes to **remAll** so that it removes all of the elements ***up to and including the first*** **e** in **L**. (We could call it **remUpto**.)

`remUpto('d','coded')` ➡ `'ed'`

If e is not in L, remUpto should remove everything...

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig,
        False otherwise. Both are strings.
    """
    if s == '':
        return True
    elif
```

Challenge...

3 Write the other cases needed for **subseq**...

```
subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
```

Name(s):

Quiz

cs5 hrs last week

Algorithm design

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + re
    else:
        return remAll(e,L[1:])
```

1

Change **remAll** so that it removes only ***one*** **e** from **L**. (We could call it **remOne**.)

remOne(8,[7,8,9,8]) ➔ **[7,9,8]**

Hint: remove one thing for **remOne**!

2

Make *more* changes to **remAll** so that it removes all of the elements ***up to and including the first*** **e** in **L**. (We could call it **remUpto**.)

remUpto('d','coded') ➔ **'ed'**

If e is not in L, remUpto should remove everything...

Hint: remove one ***more*** thing for **remUpto**!

Hint: In both cases, what's needed is *mostly crossing stuff out*! *What stuff?*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig,
        False otherwise. Both are strings.
    """
    if s == '':
        return True
    elif
```

Challenge...

3 Write the other cases needed for **subseq**...

```
subseq('alg','magical')
lse

subseq('alg','twasbrillig')
True
```

Hint: you'll need 3-4 cases total for **subseq**.

from **remAll** to **remOne**

Hint: remove one thing for **remOne**!

```
            One
def rem~~All~~( e, L ):
    """ returns seq. L with all e's rmovd
    """
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return  L[0:1] + remAll( e, L[1:] )

    else:
        return  remAll( e, L[1:] )
```

`remOne(8,[7,8,9,8])` → `[7,9,8]`

`remOne('d','coded')` → `'coed'`

from **remOne** to **remUpto**

Hint: remove one ***more*** thing for **remUpto**!

```
       Upto
def rem~~One~~( e, L ):
    """ returns seq. L with one e rmoved
    """
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return  L[0:1] + remOne( e, L[1:] )

    else:
        return  L[1:]
```

remUpto(8,[7,8,9,8]) → [9,8]

remUpto('d','coded') → 'ed'

Subseq ~ *trying (coding) it out...*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig;
        False otherwise. Both are strings.
    """
    if s == '':
        return True
```

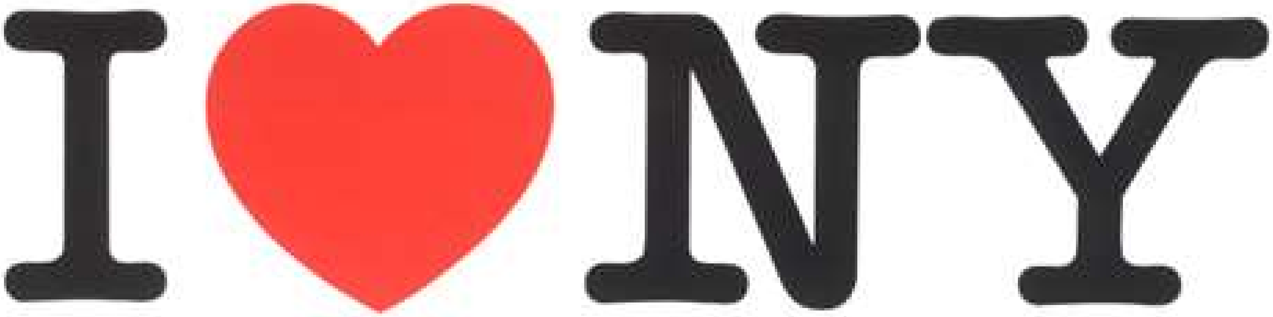
Base case(s)

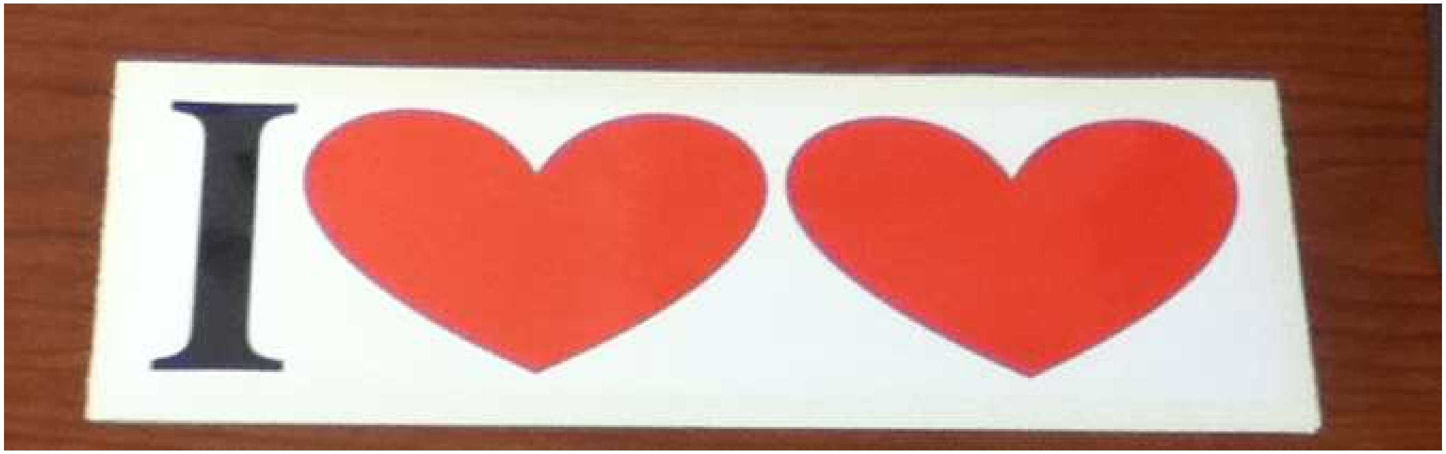
Recursive step(s)

```
subseq('alg','magical')
False
```

```
subseq('alg','twasbrillig')
True
```

I❤NY


I ♥ NY


I ♥ ♥

I ♥ NY
I ♥ I ♥ I ♥ I ♥ I ♥
I ♥ ♥

DX
I ♥ I ♥

Subseq ~ *trying it out…*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig;
        False otherwise. Both are strings.
    """
    if s == '':
        return True
```

Base case(s)

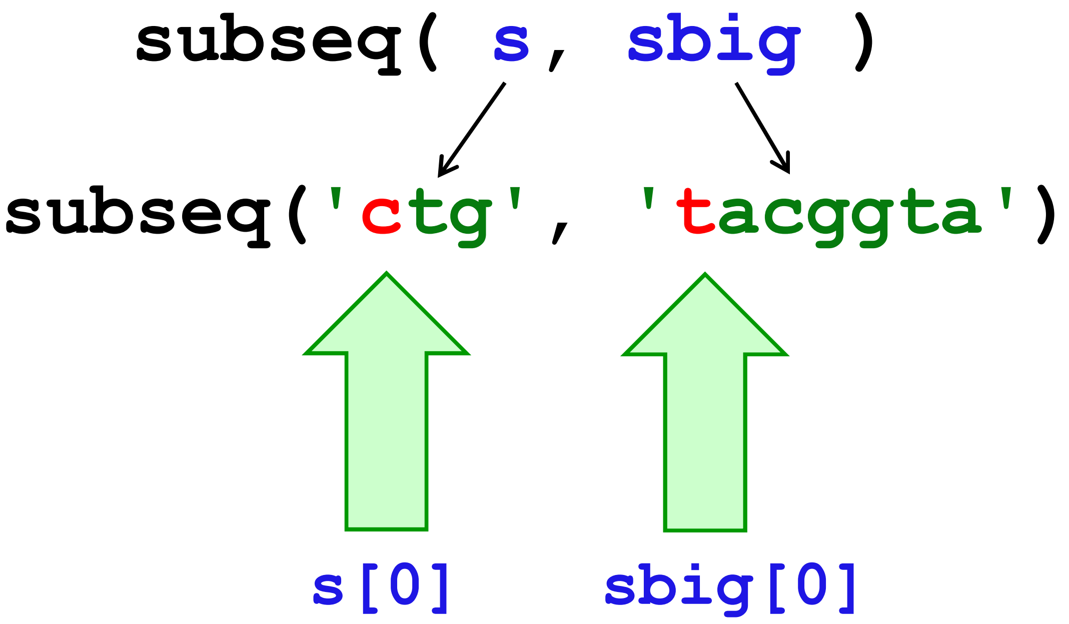
Recursive step(s)

```
subseq('alg','magical')
False
```

```
subseq('alg','twasbrillig')
True
```

I❤NY

Subseq ~ *thinking* it out...



```
subseq( s, sbig )

subseq('ctg', 'tacggta')
```

`s[0]`   `sbig[0]`

Use it!

What is a small (initial) piece of the problem?

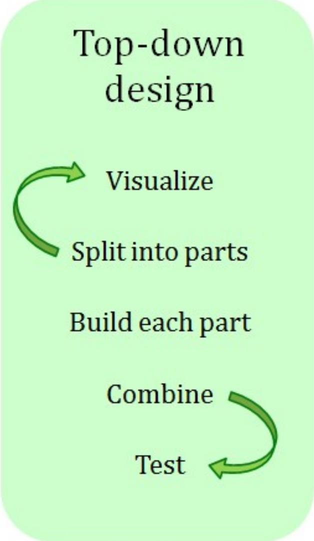
How would we describe it in terms of the inputs?

- or -

Lose it!

What is left after handling this piece?

Are there other functions we will need?



Top-down design

Visualize

Split into parts

Build each part

Combine

Test

Subseq ~ *coding it out...*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig;
        False otherwise. Both are strings.
    """
    if s == '':
        return True

    elif s[0] not in sbig:
        return False

    elif s[0] == sbig[0]:
        return subseq( s[1:], sbig[1:] )

    else:
        return subseq( s[0:], sbig[1:] )
```

it

Base case(s)

Recursive step(s)

Where are the *useit* and *loseit* here?

Name(s):

Quiz

cs5 hrs last week

Algorithm design

```
def remAll( e, L ):
    """ removes all e's from L """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remAll(e,L[1:])
    else:
        return remAll(e,L[1:])
```

1

Change **`remAll`** so that it removes only ***one*** **`e`** from **`L`**. (We could call it **`remOne`**.)

`remOne(8,[7,8,9,8])` ➡ `[7,9,8]`

2

Make *more* changes to **`remAll`** so that it removes all of the elements ***up to and including the first*** **`e`** in **`L`**. (We could call it **`remUpto`**.)

`remUpto('d','coded')` ➡ `'ed'`

If e is not in L, remUpto should remove everything...

Hint: In both cases, what's needed is *mostly crossing stuff out!* *What stuff?*

```
def subseq( s, sbig ):
    """ returns True if s is a subseq. of sbig,
        False otherwise. Both are strings.
    """
    if s == '':
        return True
    elif
```

Challenge...

3

Write the other cases needed for **`subseq`**...

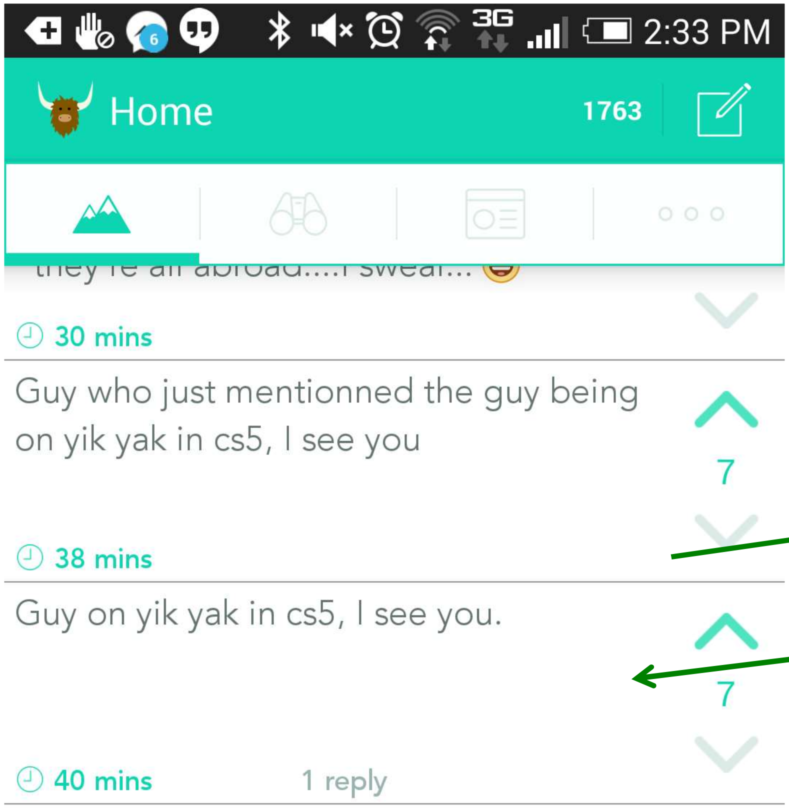
```
subseq('alg','magical')
False

subseq('alg','twasbrillig')
True
```

Pass those dormwards...

Design ~ *(code)*

That's it. ***Algorithmic expression*** ~
it's what CSers (think they) do.

can take some *"getting used to"* ... ?



they're all abroad....I swear... 😄

30 mins

Guy who just mentionned the guy being on yik yak in cs5, I see you

7

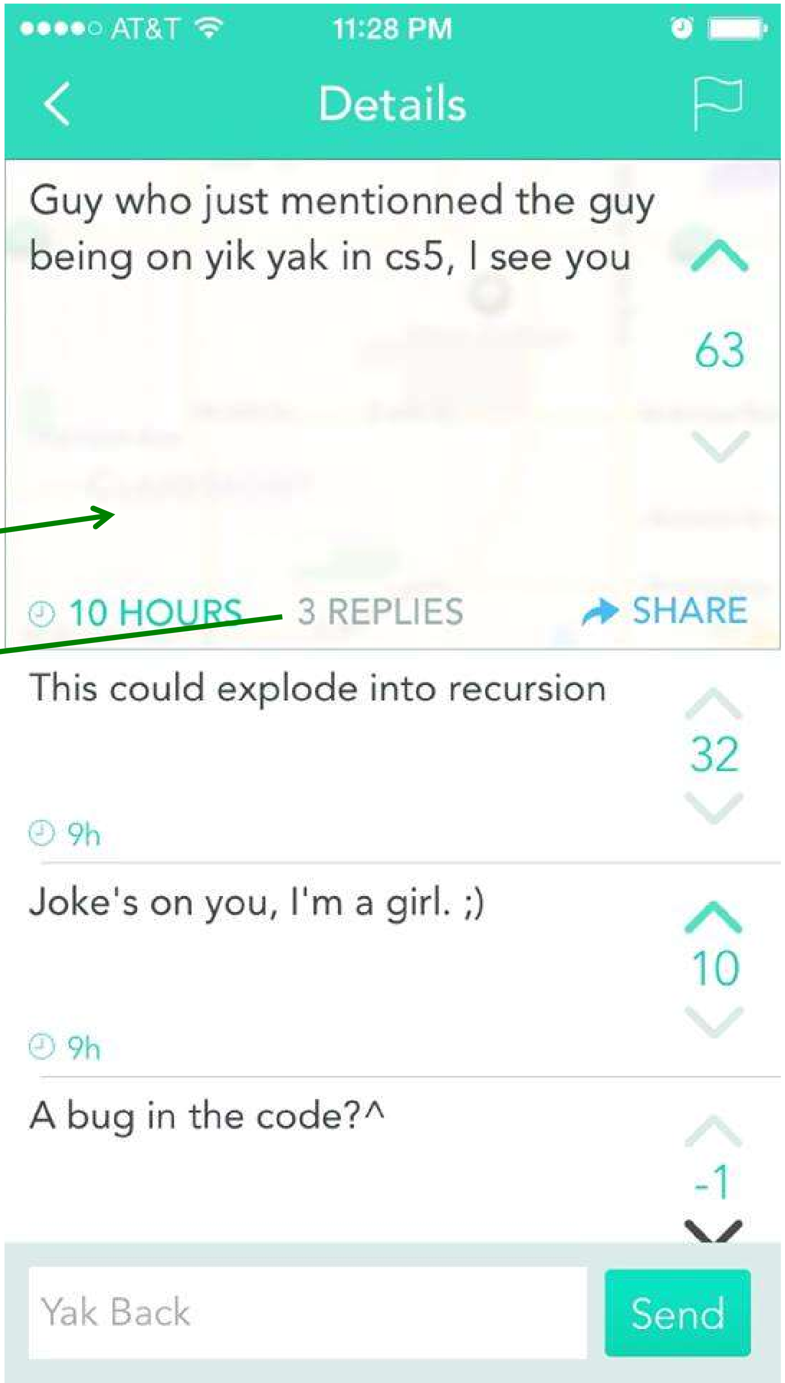
38 mins

Guy on yik yak in cs5, I see you.

7

40 mins 1 reply

?



Guy who just mentionned the guy being on yik yak in cs5, I see you

63

10 HOURS 3 REPLIES SHARE

This could explode into recursion

32

9h

Joke's on you, I'm a girl. ;)

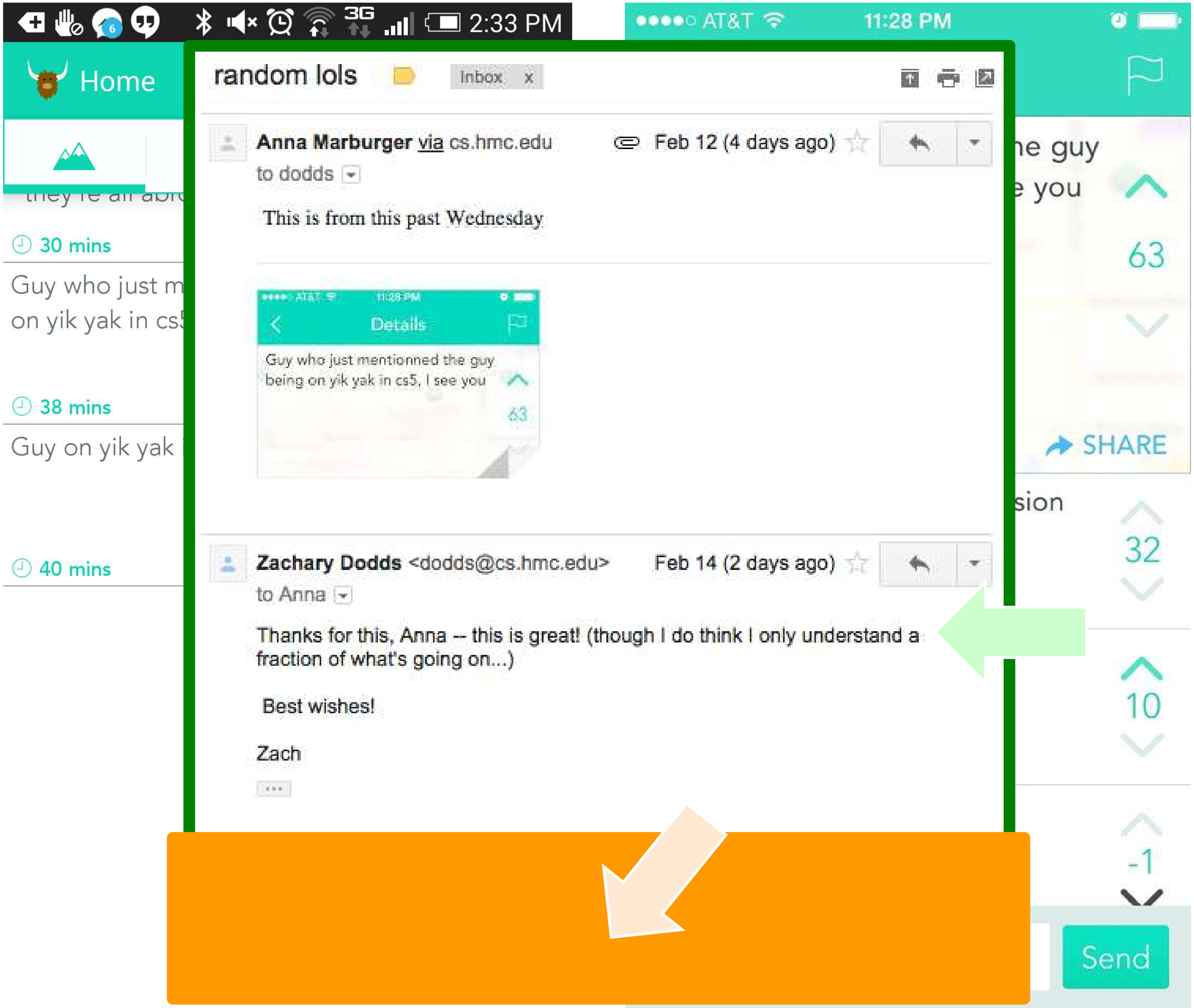
10

9h

A bug in the code?^

-1

... at this time in prior CS5 ...



random lols

Anna Marburger via cs.hmc.edu — Feb 12 (4 days ago)
to dodds

This is from this past Wednesday

Details

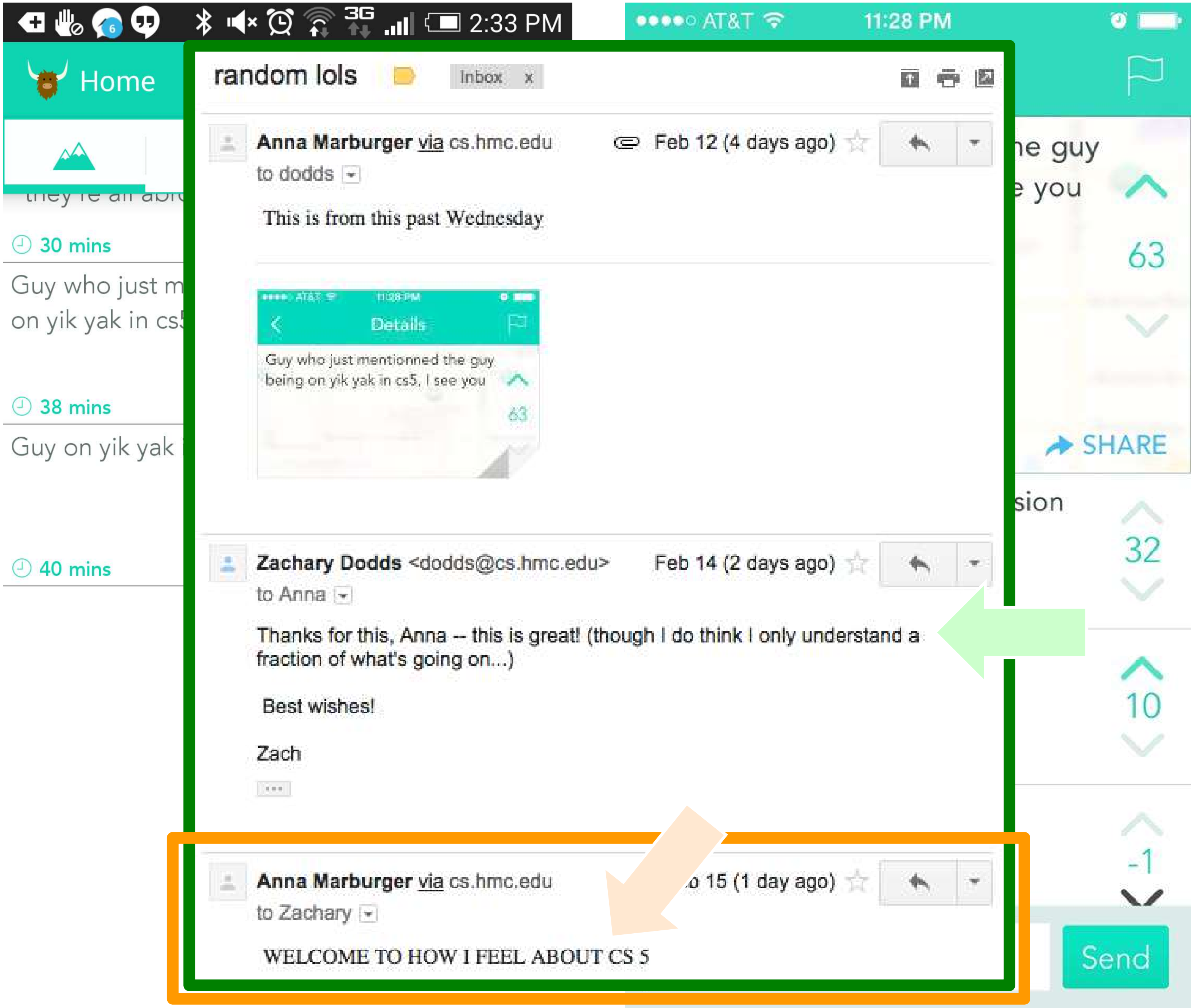
Guy who just mentionned the guy being on yik yak in cs5, I see you

Zachary Dodds <dodds@cs.hmc.edu> — Feb 14 (2 days ago)
to Anna

Thanks for this, Anna -- this is great! (though I do think I only understand a fraction of what's going on...)

Best wishes!

Zach



random lols

Inbox x

Anna Marburger via cs.hmc.edu — Feb 12 (4 days ago)
to dodds

This is from this past Wednesday

Details

Guy who just mentioned the guy being on yik yak in cs5, I see you

63

Zachary Dodds <dodds@cs.hmc.edu> — Feb 14 (2 days ago)
to Anna

Thanks for this, Anna -- this is great! (though I do think I only understand a fraction of what's going on...)

Best wishes!

Zach

Anna Marburger via cs.hmc.edu — o 15 (1 day ago)
to Zachary

WELCOME TO HOW I FEEL ABOUT CS 5

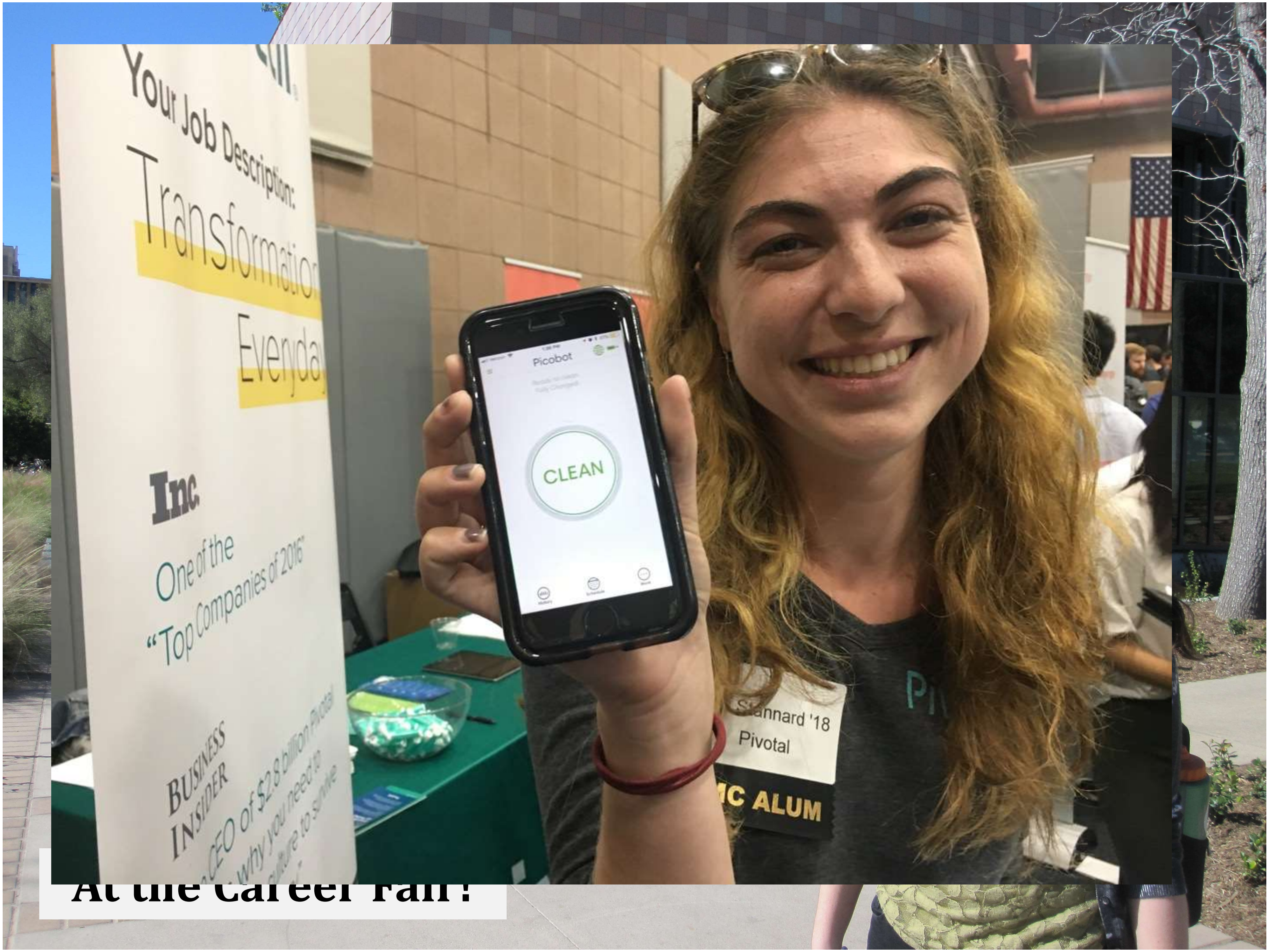
Design ~ *(code)*

That's it. ***Algorithmic expression ~***
it's what CSers (think they) do.

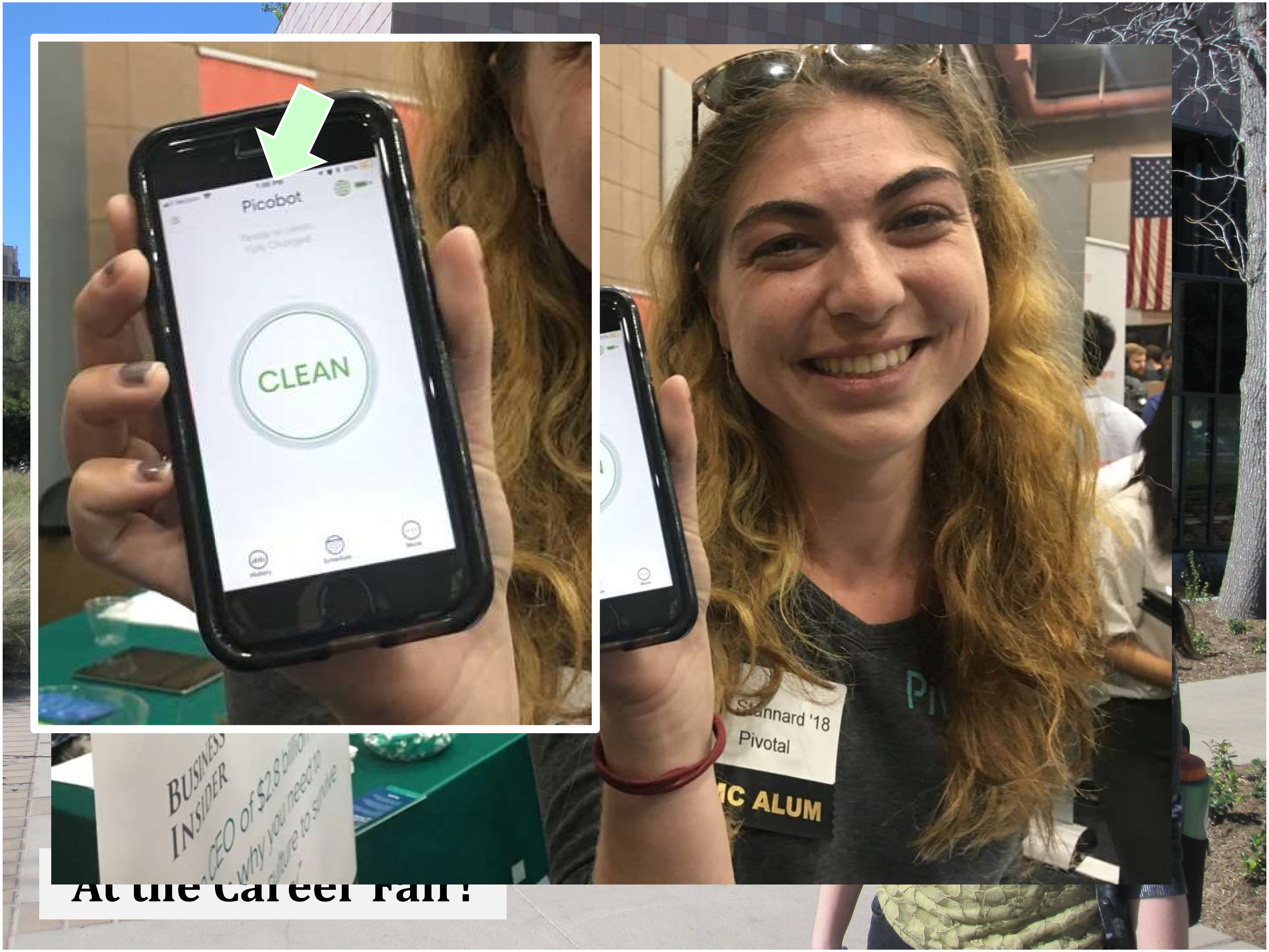
can take some *"getting used to"* ... ? ⬇

million "troubles" vs. million "triumphs"

Teal… !

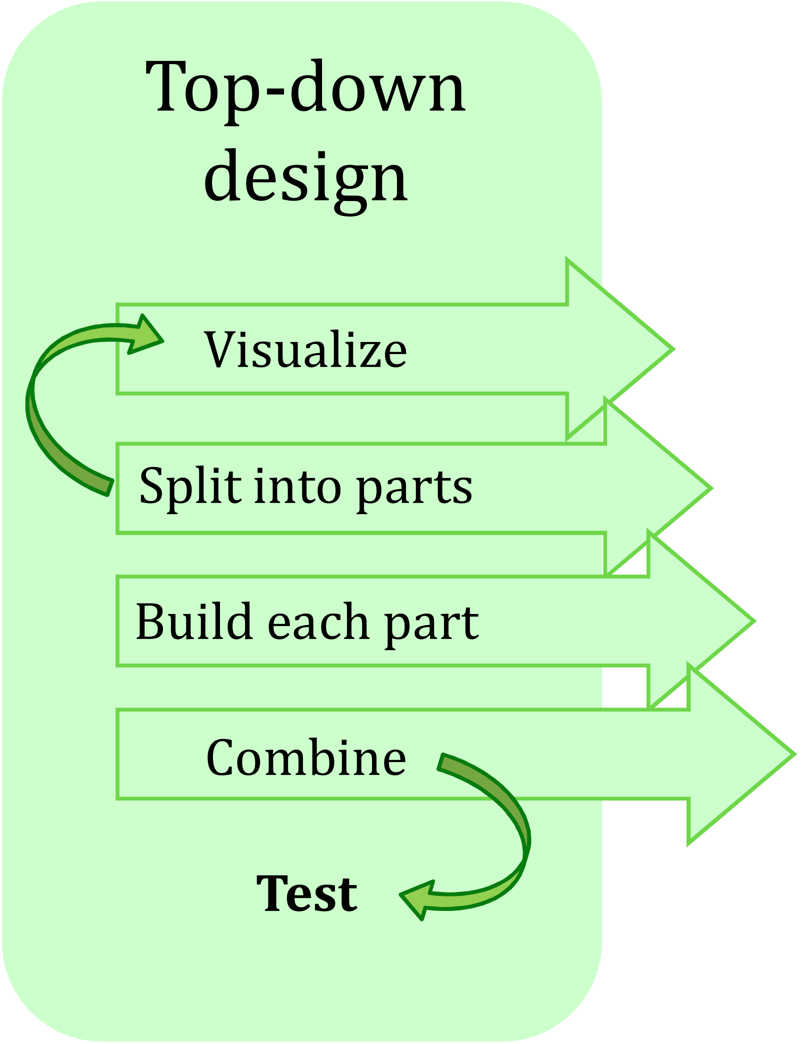


At the Career Fair!



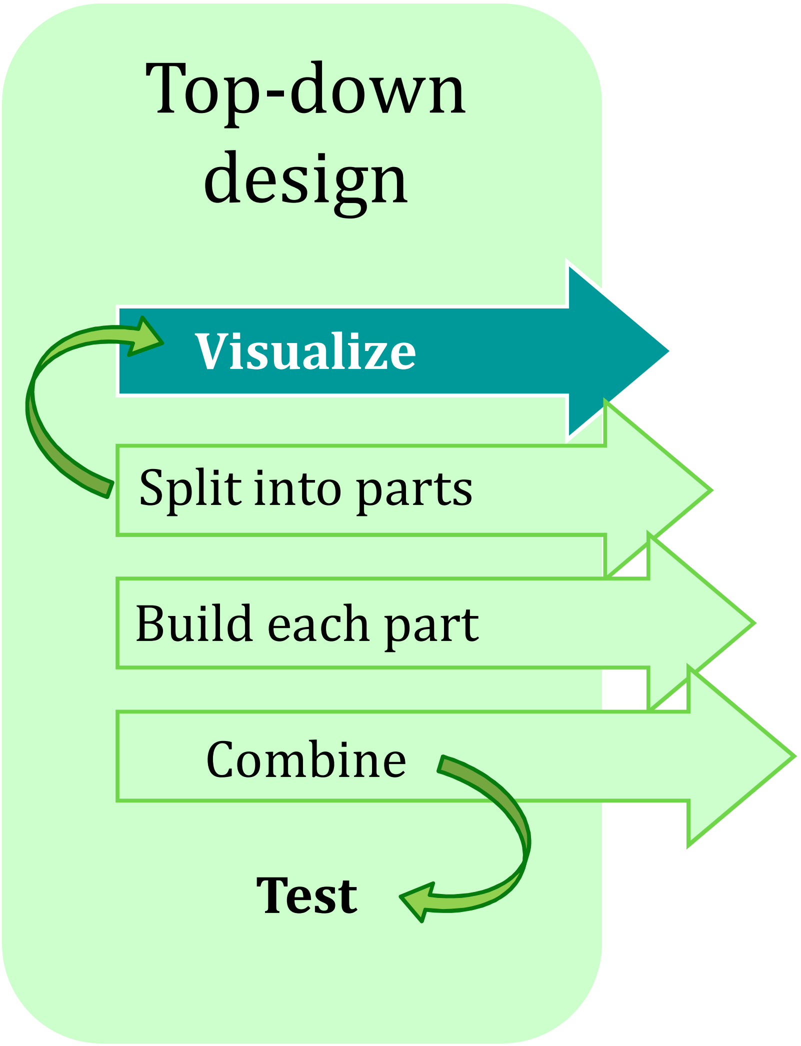
At the Career Fair!

*What's the **problem**?!*



*Which **one** of these steps is the most important?*

What's the problem?!



Top-down design

Visualize

Split into parts

Build each part

Combine

Test

understanding
what the problem demands!!

I want some ***examples****!*

hw3pr2: *use it or lose it*

Longest **C**ommon **S**ubsequence	LCS(S, T)
Jotto **Score** counting	jscore(s1, s2)
binary **l**ist and **gen**eral list **sort**ing	blsort(L), gensort(L)
exact_change making	exact_change(t, L)

hw3pr2: *use it or lose it*

Longest **C**ommon **S**ubsequence

LCS(S, T)

'HUMAN'

'CHIMPANZEE'

'CGCTGAGCTAGGCA...'

'ATCCTAGGTAACTG...'

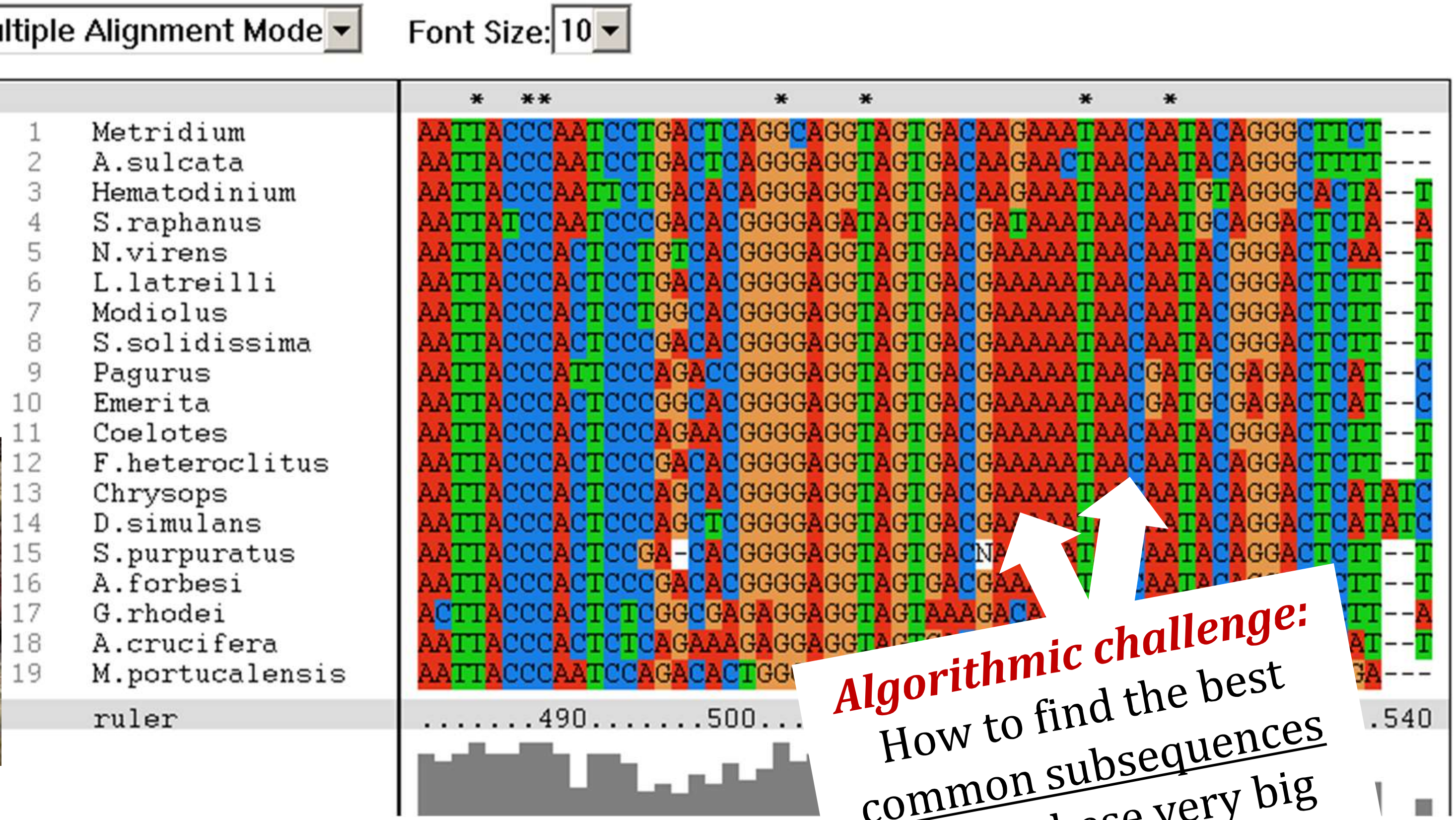
$+10^9$ more

Eye oneder if this haz other aplications?

design of what? ***Code?*** ***Algorithms!***

Why LCS?

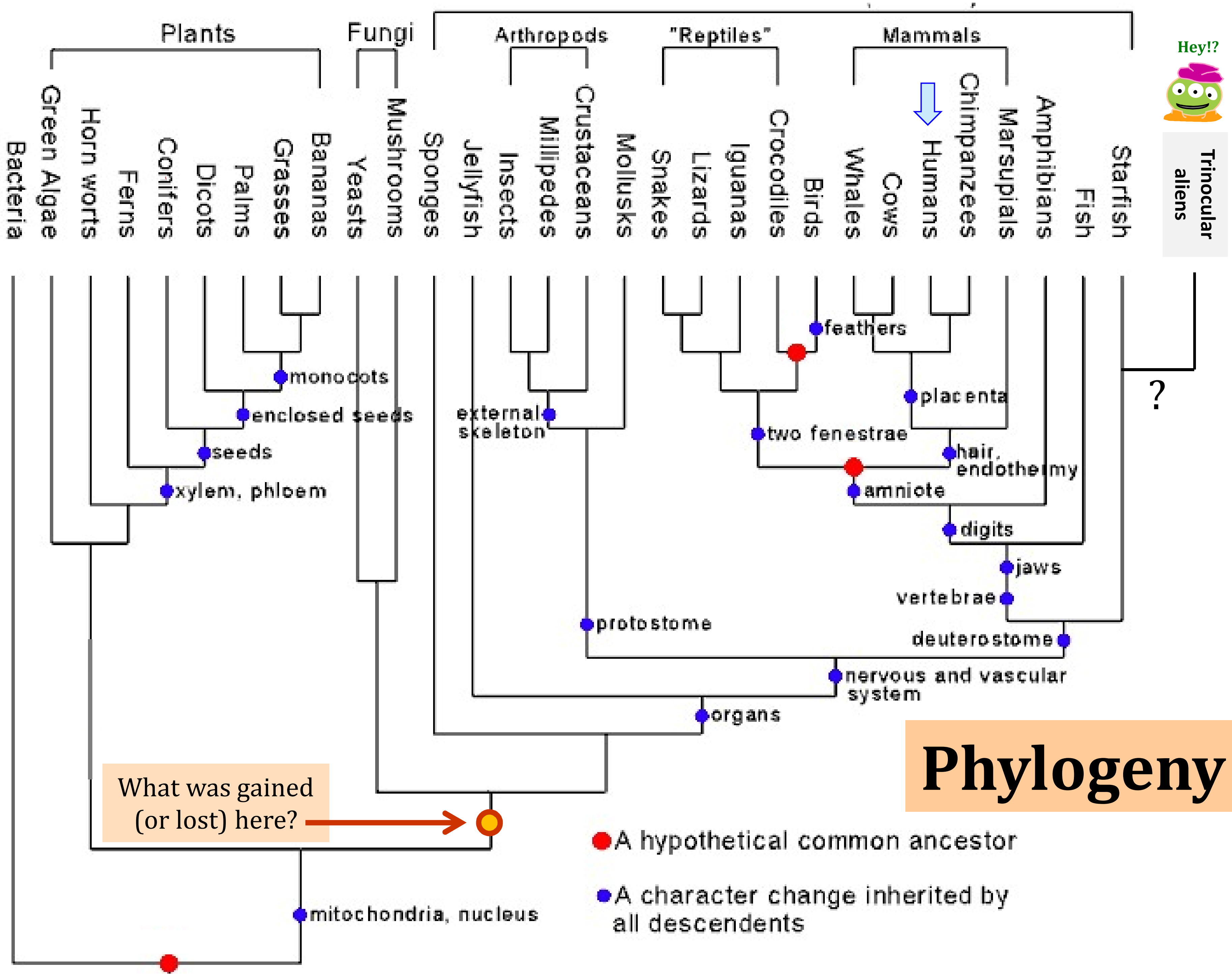
Screenshot from the ClustalX multiple subsequence alignment tool...



```
                           *  **             *    *            *    *
 1  Metridium           AATTACCCAATCCTGACTCAGGCAGGTAGTGACAAGAAATAACAATACAGGGCTTCT---
 2  A.sulcata           AATTACCCAATCCTGACTCAGGGAGGTAGTGACAAGAACTAACAATACAGGGCTTTT---
 3  Hematodinium        AATTACCCAATTCTGACACAGGGAGGTAGTGACAAGAAATAACAATGTAGGGCACTA--T
 4  S.raphanus          AATTATCCAATCCCGACACGGGGAGATAGTGACGATAAATAACAATGCAGGACTCTA--A
 5  N.virens            AATTACCCACTCCTGTCACGGGGAGGTAGTGACGAAAAATAACAATACGGGACTCAA--T
 6  L.latreilli         AATTACCCACTCCTGACACGGGGAGGTAGTGACGAAAAATAACAATACGGGACTCTT--T
 7  Modiolus            AATTACCCACTCCTGGCACGGGGAGGTAGTGACGAAAAATAACAATACGGGACTCTT--T
 8  S.solidissima       AATTACCCACTCCCGACACGGGGAGGTAGTGACGAAAAATAACAATACGGGACTCTT--T
 9  Pagurus             AATTACCCATTCCCAGACCGGGGAGGTAGTGACGAAAAATAACGATGCGAGACTCAT--C
10  Emerita             AATTACCCACTCCCGGCACGGGGAGGTAGTGACGAAAAATAACGATGCGAGACTCAT--C
11  Coelotes            AATTACCCACTCCCAGAACGGGGAGGTAGTGACGAAAAATAACAATACGGGACTCTT--T
12  F.heteroclitus      AATTACCCACTCCCGACACGGGGAGGTAGTGACGAAAAATAACAATACAGGACTCTT--T
13  Chrysops            AATTACCCACTCCCAGCACGGGGAGGTAGTGACGAAAAATA  AATACAGGACTCATATC
14  D.simulans          AATTACCCACTCCCAGCTCGGGGAGGTAGTGACGA         ATACAGGACTCATATC
15  S.purpuratus        AATTACCCACTCCGA-CACGGGGAGGTAGTGACNA   AT   AATACAGGACTCTT--T
16  A.forbesi           AATTACCCACTCCCGACACGGGGAGGTAGTGACGAA   T   AATA       CTT--T
17  G.rhodei            ACTTACCCACTCTCGGCGAGAGGAGGTAGTAAAGACA                  TT--A
18  A.crucifera         AATTACCCACTCTCAGAAAGAGGAGGTA                           AT--T
19  M.portucalensis     AATTACCCAATCCAGACACTGG                                 GA---

    ruler               .......490.......500...                                 .540
```

Algorithmic challenge: How to find the best <u>common subsequences</u> among these very big genome strings ?!?

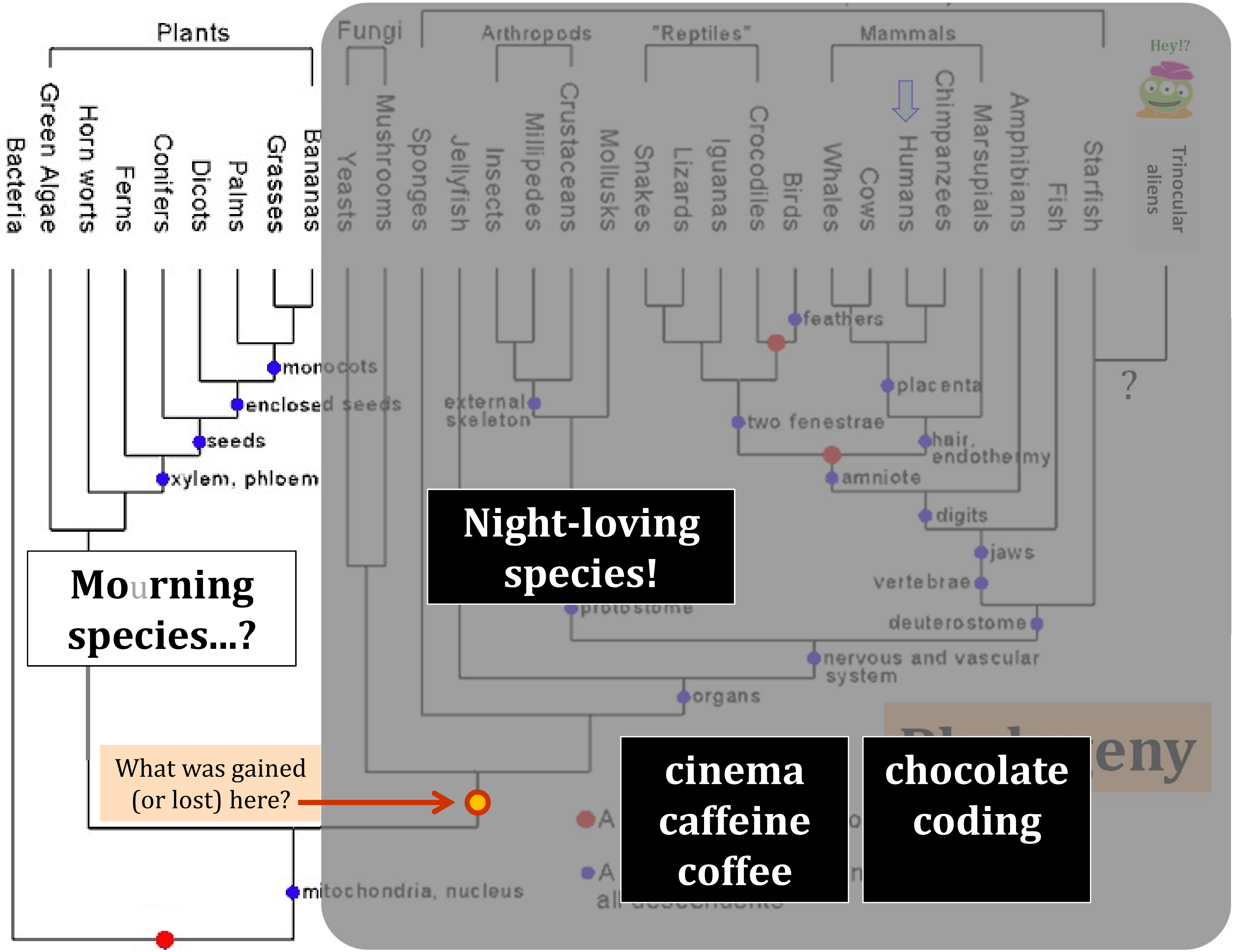
Phylogeny



What was gained (or lost) here?

● A hypothetical common ancestor

● A character change inherited by all descendents



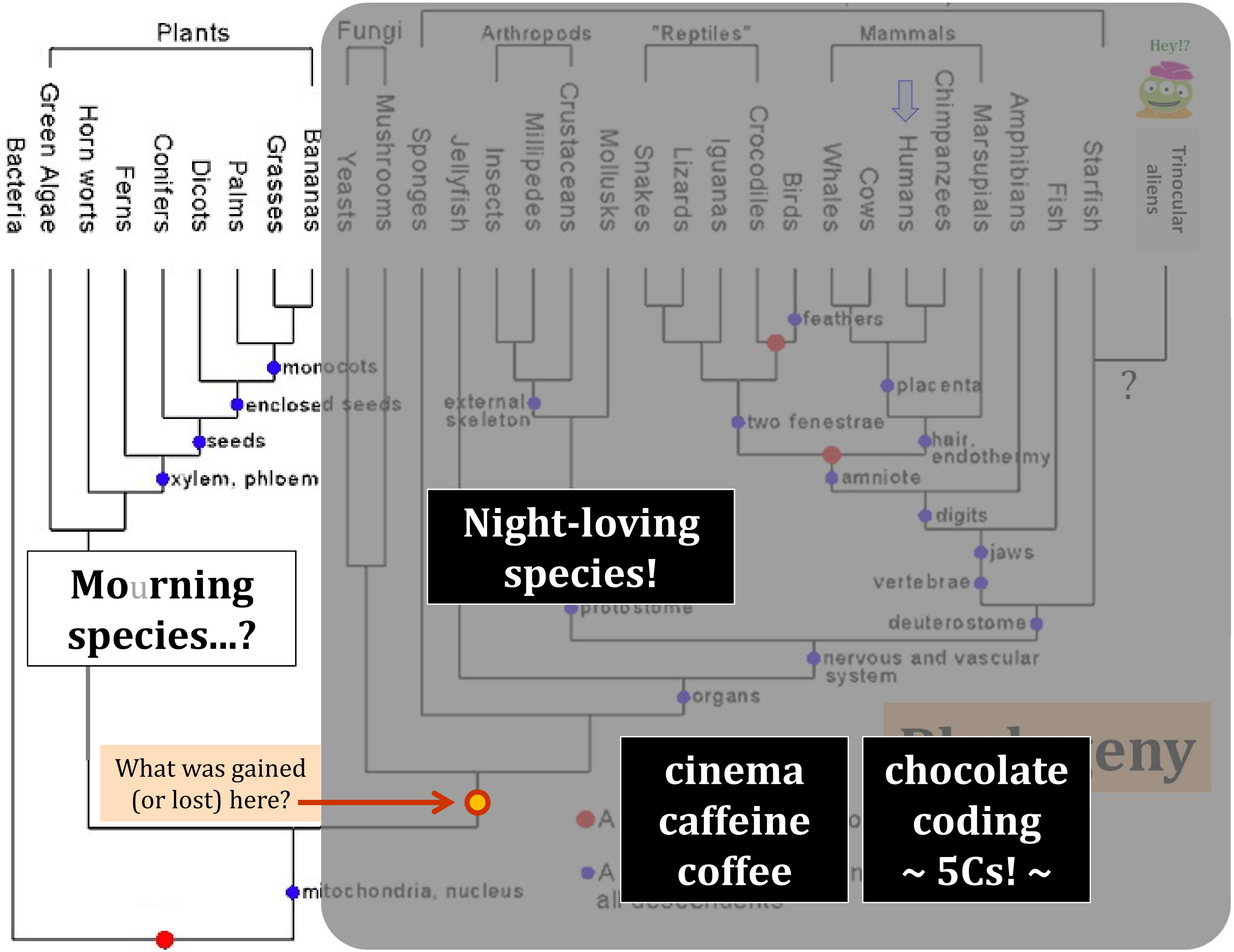
Night-loving species!

Mourning species...?

What was gained (or lost) here?

cinema
caffeine
coffee

chocolate
coding



Mourning species...?

Night-loving species!

What was gained (or lost) here?

cinema
caffeine
coffee

chocolate
coding
~ 5Cs! ~

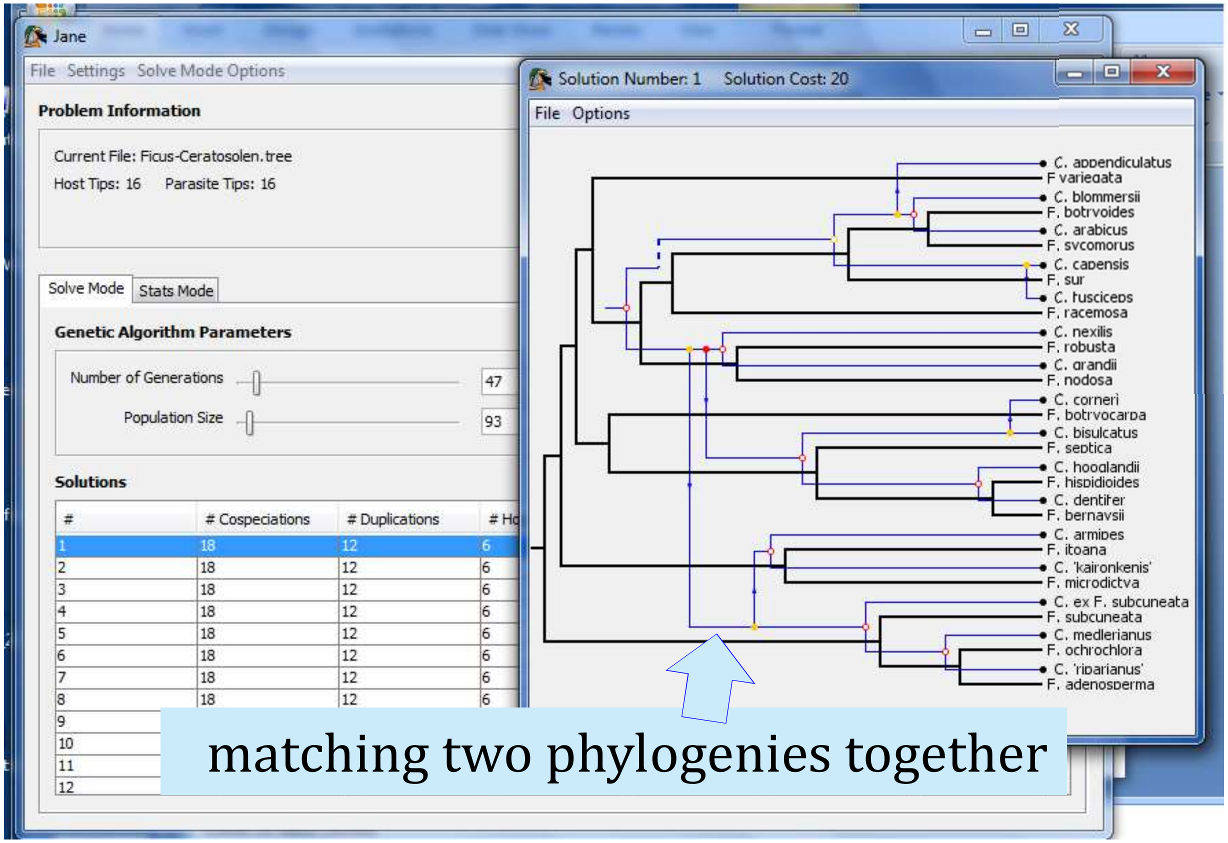
Subsequences @ 5Cs

Jane

host: figs

parasites: wasps

together!



Jane

Problem Information

Current File: Ficus-Ceratosolen.tree

Host Tips: 16 Parasite Tips: 16

Genetic Algorithm Parameters

Number of Generations 47

Population Size 93

Solutions

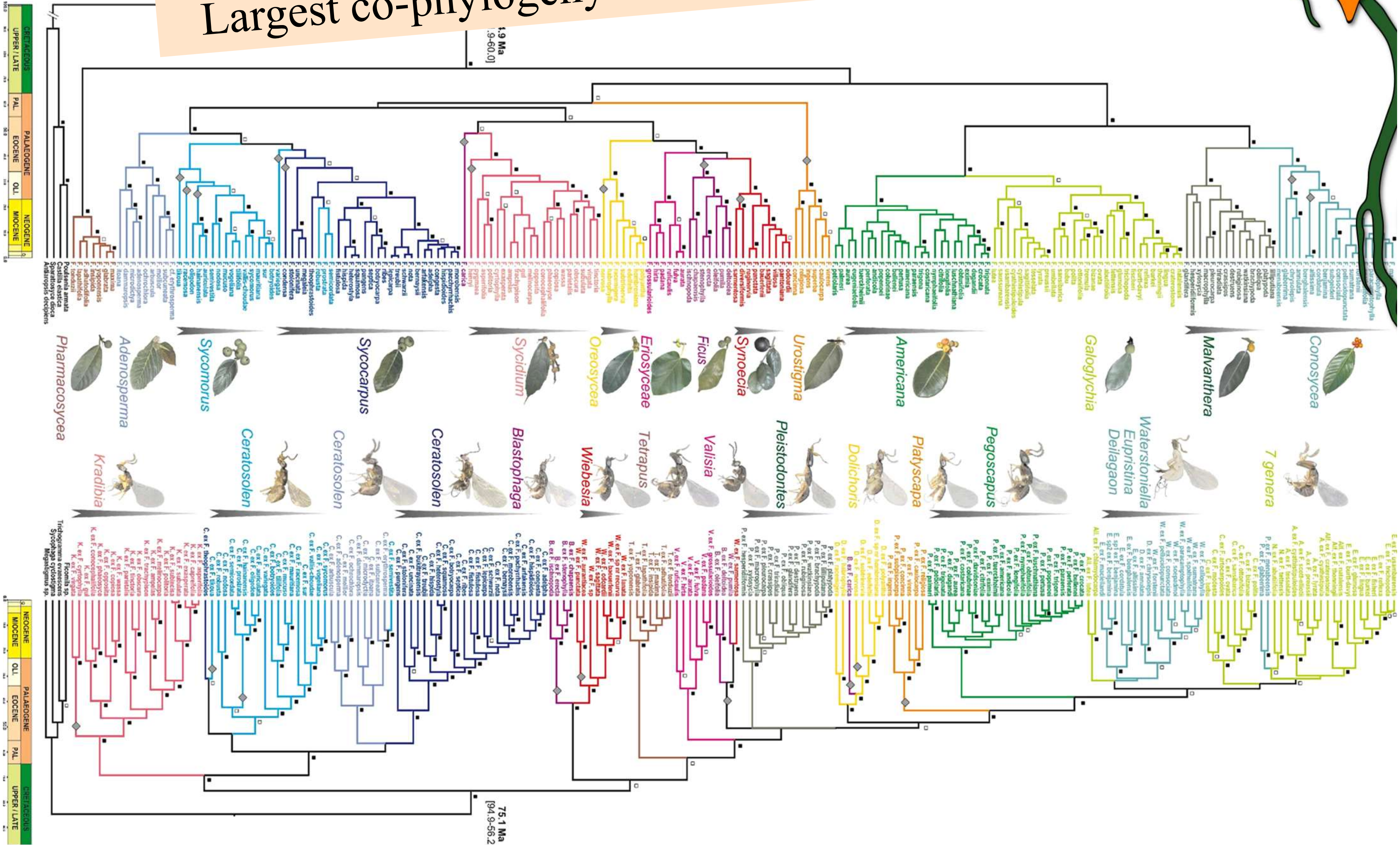
#	# Cospeciations	# Duplications	# Ho
1	18	12	6
2	18	12	6
3	18	12	6
4	18	12	6
5	18	12	6
6	18	12	6
7	18	12	6
8	18	12	6
9			
10			
11			
12			

Solution Number: 1 Solution Cost: 20

matching two phylogenies together

Jane's source data: 100s of species, 6 continents …

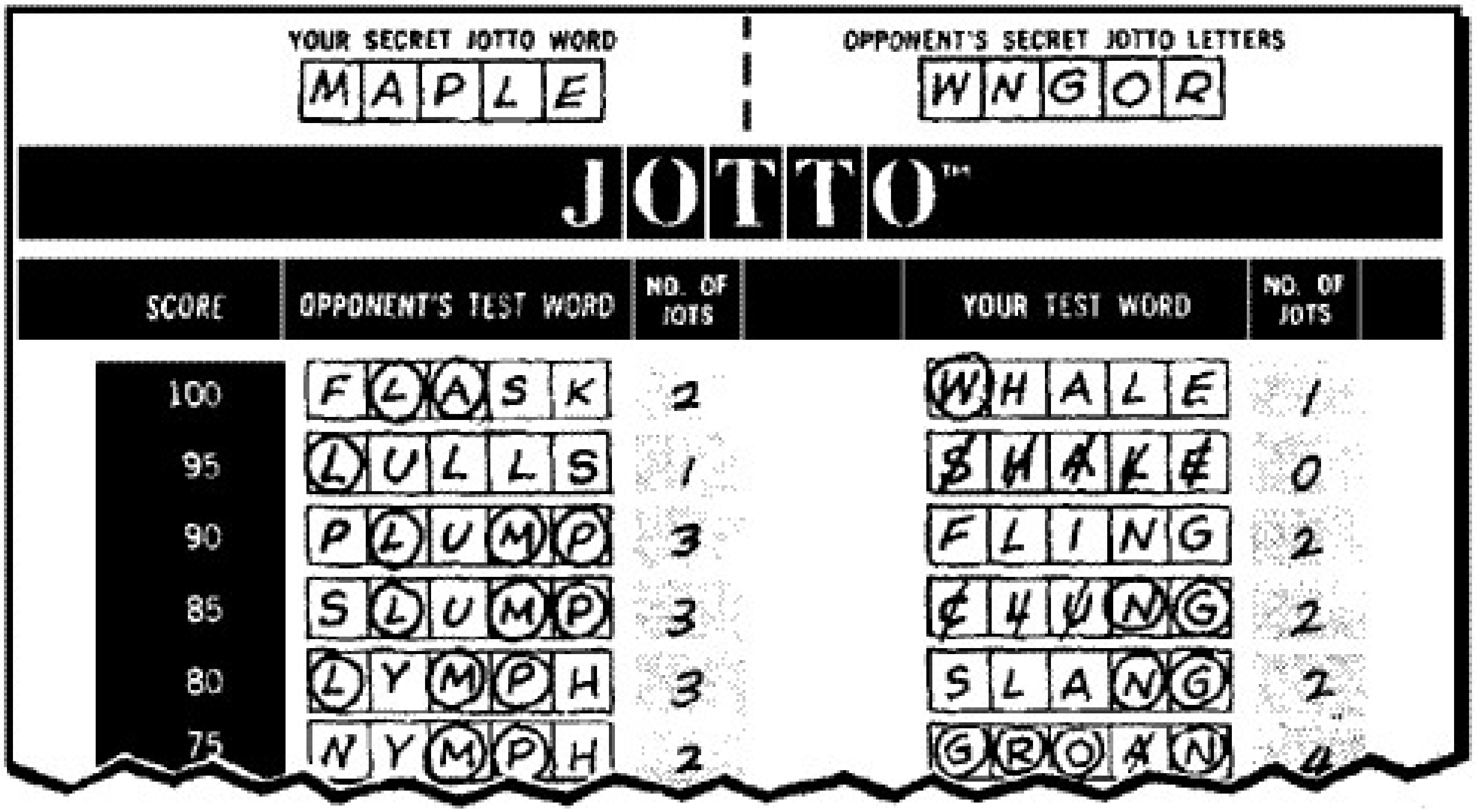
Largest co-phylogeny ever computed



also in hw3pr2: *Jotto* !

a word-guessing game…

jscore(S, T)



YOUR SECRET JOTTO WORD: M A P L E

OPPONENT'S SECRET JOTTO LETTERS: W N G O R

JOTTO™

SCORE	OPPONENT'S TEST WORD	NO. OF JOTS		YOUR TEST WORD	NO. OF JOTS
100	FLASK	2		WHALE	1
95	LULLS	1		~~SHAKE~~	0
90	PLUMP	3		FLING	2
85	SLUMP	3		~~CLU~~NG	2
80	LYMPH	3		SLANG	2
75	NYMPH	2		GRO~~A~~N	4

jscore

"Jotto scoring"

Let's try it!

These are two cute

'robot'

'otter'

jscore('robot', 'otter') →

jscore(S, T)

in general...

also in hw3pr2: **sort + exact_change**

sort([42,5,7] **)** →

sort([42,7] **)** →

sort([42] **)** →

returns an ascending list

---

returns **True** or **False**

exact_change(42, [25,30,2,5] **)** →

exact_change(42, [25,30,2,15] **)** →

should return the jotto score
for any strings **s1** and **s2**

jscore(s1,s2)

jscore('robot', 'otter') → 3

jscore('geese', 'seems') → 3

jscore('fluff', 'lulls') → 2

jscore('pears', 'diner') →

jscore('xylyl', 'slyly') →

Extra! Which of these 10 is the
cruellest hidden jotto word?

Use it!

Lose it!

???

should return a new list that is
the sorted version of the input **L**

sort(L)

sort([42,5,7]) → **[5,7,42]**

sort([42,7]) → **[7,42]**

sort([42]) → **[42]**

sort([ ]) →

blsort([1,0,1]) →

binary-list sort:
same as sort, but all
of the #s are 0 or 1

don't write
any code
for these...

???

???

should return the Longest Common
Subsequence of strings **S** and **T**

LCS(S,T)

LCS('ctga', 'tagca' **)** → 'tga'

LCS('tga', 'taacg' **)** → 'ta' (or 'tg')

LCS('tga', 'a' **)** →

LCS('gattaca', 'ctctgcgat'**)** →

do try the
examples +
brainstorm

???

Brainstorm algorithms for these problems. What **helper functions???** might help for each...

returns True if ***any*** subset of elements in L
add up to t; returns False otherwise

exact_change(t,L)

exact_change(42, **[25,30,2,5])** → **False**

exact_change(42, **[22,16,3,2,17])** →

exact_change(42, **[18,21,22])** →

exact_change(42, **[40,17,1,7])** →

exact_change(20, **[16,3,2,17])** →

???

should return the jotto score for any strings **s1** and **s2**

`jscore(s1,s2)`

jscore('robot', 'otter') → 3

jscore('geese', 'seems') → 3

jscore('fluff', 'lulls') → 2

jscore('pears', 'diner') → 2

jscore('xylyl', 'slyly') → 4

Extra! Which of these 10 is the *cruellest* hidden jotto word?

Use it!

Lose it!

???

don't write any code for these...

should return a new list that is the sorted version of the input **L**

`sort( L )`

sort([42,5,7]) → [5,7,42]

sort([42,7]) → [7,42]

sort([42]) → [42]

sort([ ]) → [ ]

blsort([1,0,1]) → [0,1,1]

binary-list sort:
same as sort, but all
of the #s are 0 or 1

???

Use it!

???

do try the examples + brainstorm

should return the Longest Common Subsequence of strings **S** and **T**

`LCS(S,T)`

LCS('ctga', 'tagca') → 'tga'

LCS('tga', 'taacg') → 'ta' (or 'tg')

LCS('tga', 'a') → 'a'

LCS('gattaca', 'ctctgcgat') → 'ttca'

4 chars

Use it!

Lose it!

Lose it!

???

Brainstorm algorithms for these problems. What helper functions??? might help for each...

returns True if ***any*** subset of elements in L add up to t; returns False otherwise

`exact_change(t,L)`

exact_change(42, [25,30,2,5]) → False

exact_change(42, [22,16,3,2,17]) → True

exact_change(42, [18,21,22]) → False

exact_change(42, [40,17,1,7]) → False

exact_change(20, [16,3,2,17]) → True

Use it!

Lose it!

???

should return the jotto score
for any strings **s1** and **s2**

jscore(s1,s2)

jscore('robot', 'otter') → 3

jscore('geese', 'seems') → 3

jscore('fluff', 'lulls') → 2

jscore('pears', 'diner') → 2

jscore('xylyl', 'slyly') → 4

Extra! Which of these 10 is the
cruellest hidden jotto word?

Use it!

Lose it!

remOne

don't write
any code
for these...

should return a new list that is
the sorted version of the input **L**

sort(L)

sort([42,5,7]) → [5,7,42]

sort([42,7]) → [7,42]

sort([42]) → [42]

sort([ ]) → [ ]

blsort([1,0,1]) → [0,1,1]

binary-list sort:
same as sort, but all
of the #s are 0 or 1

min

Use it!

remOne

do try the
examples +
brainstorm

should return the Longest Common
Subsequence of strings **S** and **T**

LCS(S,T)

LCS('ctga', 'tagca') → 'tga'

LCS('tga', 'taacg') → 'ta' (or 'tg')

LCS('tga', 'a') → 'a'

LCS('gattaca', 'ctctgcgat') → 'ttca'

4 chars

Use it!

Lose it!

Lose it!

only recursion
here...

Brainstorm algorithms for these problems -- what **helper functions???** might help for each?

returns True if ***any*** subset of elements in L
add up to t; returns False otherwise

exact_change(t,L)

exact_change(42, [25,30,2,5]) → False

exact_change(42, [22,16,3,2,17]) → True

exact_change(42, [18,21,22]) → False

exact_change(42, [40,17,1,7]) → False

exact_change(20, [16,3,2,17]) → True

Use it!

Lose it!

... and
here

```
decipher( 'Weet bksa ed xecumeha 3!' )
```

kxn rkfo k qbokd goouoxn ...

```
decipher( 'Weet bksa ed xecumeha 3!' )
```

Good luck on homework 3!

kxn rkfo k qbokd goouoxn ...

and have a great weekend ...