**Algorithms**

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

**HW 4**

Hw #4 due **Monday, 11:59**

*Sound Lab!*

*Several algorithms...*

**Office Hrs.**

Today, 2-4, HMC's LAC lab...

Edward Frenkel

*Is the Universe a Simulation?*

**FEB. 14, 2014**
We saw you in Lab4...
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' )
BR 5 Snczx

**Algorithms**

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

**HW 4**

Hw #4 due **Monday, 11:59**

*Sound Lab!*

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

**Algorithms**

HW 3

Hw #3 due **Monday, 1:**
Sound Lab!
Several algorithms.

Edward Frenkel

Love & Math
The Heart of Hidden Reality

"If you're not a mathematician this book might make you want to become one."
—NASSIM NICHOLAS TALEB, author of *The Black Swan*
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

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

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

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

model for this problem: transcribe from hw#2

```python
>>> encipher('Hu lkbjhap vu 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.'

>>> 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!')

>>> decipher('gv vw dtwvg')

But how?```
<table>
<thead>
<tr>
<th>Decipher?</th>
<th>All possible decipherings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategies?</td>
<td></td>
</tr>
<tr>
<td>Algorithms?</td>
<td></td>
</tr>
</tbody>
</table>

| gv vw dtwvg |
| hw wx euxwh |
| ix xy fvyxi |
| jy yz gwzyj |
| kz za hxazk |
| la ab iybal |
| mb bc jzcbm |
| nc cd kadc |
| od de lbedo |
| pe ef mcfep |
| qf fg ndgfq |
| rg gh oehgr |
| sh hi pfihs |
| ti ij qgjit |
| uj jk rhkju |
| vk kl slikv |
| 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 |
Measuring *Englishness*

Very English-y

- "Call me Ishmael."
- "Attack at dawn!"
- "rainbow, table, candle"
- "quadruplicity drinks procrastination"
- "the gostak distims the doses"
- "hension, framble, bardle"
- "jufict, stofwus, lictpub"
- "itehbs, rsnevtr, khbsota"
- "epadxo, nojarpn, gdxokpw"
- "h o q dedqBzdrzqrzkzc"

higher scores

quantifying "Englishness"?

Not English-y

lower scores

All of these sound good to me!
Decipher?

Strategies?

Algorithms?

"Englishness" score ~ the #-of-vowels

Score them all

max!

All possible decipherings

[4, 'la ab iybal'],
[0, 'mb bc jzcbm'],
[1, 'nc cd kadcnc'],
[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 xnpqa'],
[1, 'bq qr yorqb'],
[0, 'cr rs zpsrc'],
[1, 'ds st aqtsd'],
[4, 'et tu brute'],
[3, 'fu uv csvuf']
Decipher? All possible decipherings

Strategies?

Algorithms?

$p(l_1) \cdot p(l_2) \cdot p(l_3)$

"Englishness" based on letter-probabilities

1. $y q r b$
2. $c r s z p s r c$
3. $d s s t a q t s d$
4. $e t t u b r u t e$
5. $f u u v c s v u f$

$[6.9e-05, 'g v v w d t w v g'], [3.6e-05, 'h w w x e u x w h'], [1.4e-07, 'i x x y f v y x i'], [8.8e-11, 'j y y z g w z y j'], [7.2e-10, 'k z z a h x a z k'], [0.01503, 'l a a b i y b a l'], [3.7e-08, 'm b b c j z c b m'], [0.00524, 'n c c d k a d c n'], [0.29041, 'o d d e l b e d o'], [0.00874, 'p e e f m c f e p'], [7.3e-07, 'q f f g n d g f q'], [0.06410, 'r g g h o e h g r'], [0.11955, 's h h i p f i h s'], [3.1e-06, 't i i j q g j i t'], [1.1e-08, 'u j j k r h k j u'], [2.6e-05, 'v k k l s i l k v'], [0.00012, 'w l l m t j m l w'], [3.1e-06, 'x m m n u k n m x'], [0.02011, 'y n n o v l o n y'], [1.5e-06, 'z o o p w m p o z'], [1.9e-07, 'a p p q x n q p a'], [5.7e-08, 'b q q r y o r q b'], [0.00024, 'c r r s z p s r c'], [0.02060, 'd s s t a q t s d'], [0.45555, 'e t t u b r u t e'], [0.00011, 'f u u v c s v u f']
Decipher?

All possible decipherings

gv vv dtwvg
hw wx euxwh
ix xy fvyxi
jy yz gwzyj
kz za hxazk
la ab iybal
mb bc jzcbm
nc cd kadc
od de lbedo
pe ef mcfep
vk kl silkv
wl lm tjml
xp yr cuf
ds st aqtsd
fu uv csvuf

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

"Englishness" based on scrabble-scoring!

Strategies?

Algorithms?

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

min!
[11, 'et tu brute']
[23, 'fu uv csvuf']
Design...

design of what?

The ♥ of CS
(and CSers...)

Algorithms!
Design...

design of what?

Code?
syntax
The Economist explains

What is code?

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

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
Design...

Code?

Algorithms!

design of what?

syntax

ideas!
Algorithm Design...

\[
\text{remAll}(e,L)
\]
remove all e's from L
**Design...**

**Top-down design**

- Visualize
- Split into parts
- Build each part
- Combine
- Test

**Code Examples**

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

remove all e's from L

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

[5,7,8]

```
remAll('q','qaqqlqqiqqqiiiqeqqqns')
```

'aliiiiens'
Design...

Top-down design

- Visualize
- Split into parts: L[0] and L[1:]
- Build each part
- Combine
- Test

remAll(e, L)
remove all e's from L

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

remAll('q', 'qaqqlqqiqqiiqeqqnsq')
'aliiiens'

Use it!

Lose it!
Design...

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

Use it!

remAll(e, L)
remove all e’s from L

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

keep L[0]
+ remove e from the rest

remAll('q', 'qaqqlqqiqqiiqqiieeqqqnqs')
'aliiiens'

Lose it!

remAll('q', 'qaqqlqqiqqiiqqiieeqqqnqs')
'the rest'

remAll(e, L)
remove all e’s from L

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

keep L[0]
+ remove e from the rest

remAll('q', 'qaqqlqqiqqiiqqiieeqqqnqs')
'aliiiens'

Lose it!
Design...

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

```
remAll(e, L)
```
remove all e’s from L

```
keep L[0] + remove e from rest
```

```
remAll('q', 'aqqlqqiqqiiqeqqnqs'
```

```
drop L[0] + remove e from rest
```

Use it!

- or -

Lose it.

Test

Build each part

Combine

Visualize

Split into parts
Allie Russell, '12 speaking of roadside church signs...
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:])

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

```python
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:])
```

- **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

Top-down design

Re-Visualize in syntax!?
Design ~ code

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

```python
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:])
```

That's it.
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:])

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

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

Algorithm design

1. Change remAll so that it removes only one e from L. (We could call it remOne.)
remOne(8, [7, 8, 9, 8]) \rightarrow [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') \rightarrow 'ed'

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

Challenge...
subseq('alg', 'magical')
False
subseq('alg', 'twasbrillig')
True

Hint: In both cases, what's needed is mostly crossing stuff out. What stuff?
def remAll( e, L ):
    
    # remove 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:]


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

def remUpto( e, L ):
    
    # remove all items in L up to and including the first e in L (We could call it remUpto.)
    
    if e is not in L, remUpto should remove everything after e...
remAll insight

```python
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]

sharpening our model for where + how actions happen...
other `rem` examples...

\[
\begin{align*}
\text{remAll}(8, [7,8,9,8]) & \Rightarrow [7,9] \\
\text{remAll}('d', 'coded') & \Rightarrow 'coe'
\end{align*}
\]

\[
\begin{align*}
\text{remOne}(8, [7,8,9,8]) & \Rightarrow [7,9,8] \\
\text{remOne}('d', 'coded') & \Rightarrow 'coed'
\end{align*}
\]

\[
\begin{align*}
\text{remUpto}(8, [7,8,9,8]) & \Rightarrow [9,8] \\
\text{remUpto}('d', 'coded') & \Rightarrow 'ed'
\end{align*}
\]
Subsequences

```python
def subseq(s, sbig):
    # s is the subsequence to find (or not)
    # sbig is the bigger string in which we are looking for s
    # True or False?
```

Here there be NO dragons!

Why Are these True? or False?

- `subseq('', 'cataga')` → True
- `subseq('ctg', 'cataga')` → True
- `subseq('ctg', 'tacggta')` → False
- `subseq('aliens', 'always frighten dragons')` → True
- `subseq('trogdor', 'that dragon is gone for good')` → False
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:] )

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] in sbig:
        return subseq( s[1:], sbig )
    else:
        return False

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

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:])

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

Algorithm design

1. Change remAll so that it removes only one e from L. (We could call it 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.)
3. Write the other cases needed for subseq...

remOne(8,[7,8,9,8]) → [7,9,8]
remUpto('d','coded') → 'ed'
subseq('alg','magical')
subseq('alg','twasbrillig')

Try it out!
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:] )

Hint: remove one thing for remOne!

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] in sbig:
        return subseq( s[1:], sbig )
    else:
        return False

Hint: you'll need 3-4 cases total for subseq.
from remAll to remOne

from remAll to remOne

remOne(8, [7, 8, 9, 8]) => [7, 9, 8]     remOne('d', 'coded') => 'coed'
from remOne to remUpto

```python
def remOne( e, L ):
    """ returns seq. L with one e removed """
    if len(L) == 0:
        return L
    elif L[0] != e:
        return L[0:1] + remOne( e, L[1:] )
    else:
        return L[1:]

from remOne to remUpto

remUpto(8,[7,8,9,8]) ➞ [9,8]  
remUpto('d','coded') ➞ 'ed'
```

**Hint:** remove one more thing for `remUpto`!
def subseq(s, sbig):
    """ returns True if s is a subseq. of sbig;
    False otherwise. Both are strings. """

    if s == '':
        return True
    elif sbig == '':
        return False
    elif s[0] == sbig[0]:
        return subseq(s[1:], sbig[1:])
    else:
        return subseq(s[1:], sbig[1:])
def subseq( s, sbig ):
    """    returns True if s is a subseq. of sbig;
    False otherwise. Both are strings.
    """
    if s == '':
        return True

subseq('alg','magical')  # False
subseq('alg','twasbrillig')  # True
What is a small (initial) piece of the problem?
How would we describe it in terms of the inputs?

What is left after handling this piece?
Are there other functions we will need?
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:])
def subseq(s, sbig):
    """ returns True if s is a subseq. of sbig,
    False otherwise. Both are strings. """
    if s == '':
        return True
    elif s < sbig:
        return False
    else:
        return subseq(s[1:], sbig[1:])

if __name__ == '__main__':
    print(subseq('abc', 'ab'))
    print(subseq('abc', 'd'))
    print(subseq('abc', 'ac'))
    print(subseq('abc', 'abc'))
    print(subseq('abc', 'abcc'))

Algorithm design

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

3. Write the other cases needed for subseq. (We could call it 'ed'.)
   Challenge: ...crossing stuff out! What stuff?

Pass those in and up...

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

remOne('d', ['-coded'])
remUpto('d', ['twasbrillig'])

Name(s):

cs5 hrs last week
Design ~ (code)

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

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

Top-down design

- Visualize
- Split into parts
- Build each part
- Combine
- Test

*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

I want some examples!
hw4pr2:  use it or lose it

Longest Common Subsequence

Jotto Score counting

binary list and general list sorting

exact_change making
hw4pr2: use it or lose it

Longest Common Subsequence

LCS( S, T )

'HUMAN'

'HUMANNEX'

'CHIMPANZEE'

'CHIMPANZEEEX'

'CGCTGAGCTAGGCA...'

'CGCTGAGCTAGGCAEX'

'ATCCTAGGTAACTG...'

'ATCCTAGGTAACTGEX'

Eye oneder if this haz other aplcations?
Why LCS?

Algorithmic challenge: How to find the best common subsequences among these very big genome strings?!?
What was gained (or lost) here?

Phylogeny
What was gained (or lost) here?

Mourning species...?

Night-loving species!

caffeine cinema chocolate

Phylogeny
Subsequences @ 5Cs

host: figs

parasites: wasps

matching two phylogenies together
together!
Jane's source data: 100s of species, 6 continents ...

Largest co-phylogeny ever computed (in 2012)
also in hw4pr2: *Jotto*!

A word-guessing game...

```python
jscore(S, T)
```
also in hw4pr2: !

These are two cute

Let's try it!

'robot'  'otter'

\text{jscore}( 'robot', 'otter' ) \rightarrow

\text{jscore}( S, T ) \quad \text{in general...}
also in hw3pr2: \( \text{sort} + \text{exact\_change} \)

\[
\begin{align*}
\text{sort}( [42, 5, 7] ) & \rightarrow \\
\text{sort}( [42, 7] ) & \rightarrow \\
\text{sort}( [42] ) & \rightarrow \\
\text{exact\_change}( 42, [25, 30, 2, 5] ) & \rightarrow \\
\text{exact\_change}( 42, [25, 30, 2, 15] ) & \rightarrow \\
\end{align*}
\]

returns an ascending list
returns True or False
should return the jotto score for any strings \( s_1 \) and \( s_2 \)

\[ \text{jscore}(s_1,s_2) \]

- \( \text{jscore}(\text{'robot'}, \text{'otter'}) \rightarrow 3 \)
- \( \text{jscore}(\text{'geese'}, \text{'seems'}) \rightarrow 3 \)
- \( \text{jscore}(\text{'fluff'}, \text{'lulls'}) \rightarrow 2 \)
- \( \text{jscore}(\text{'pears'}, \text{'diner'}) \rightarrow \)
- \( \text{jscore}(\text{'xylyl'}, \text{'slyly'}) \rightarrow \)

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

\[ \text{LCS}(S,T) \]

should return the Longest Common Subsequence of strings \( S \) and \( T \)

- \( \text{LCS}(\text{'ctga'}, \text{'tagca'}) \rightarrow \text{'tga'} \)
- \( \text{LCS}(\text{'tga'}, \text{'taacg'}) \rightarrow \text{'ta'} \) (or 'tg')
- \( \text{LCS}(\text{'tga'}, \text{'a'}) \rightarrow \text{'a'} \)
- \( \text{LCS}(\text{'gattaca'}, \text{'ctctgcgat'}) \rightarrow \)

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

\[ \text{exact_change}(t,L) \]

- \( \text{exact_change}(42, [25,30,2,5]) \rightarrow \text{False} \)
- \( \text{exact_change}(42, [22,16,3,2,17]) \rightarrow \)
- \( \text{exact_change}(42, [18,21,22]) \rightarrow \)
- \( \text{exact_change}(42, [40,17,1,7]) \rightarrow \)
- \( \text{exact_change}(20, [16,3,2,17]) \rightarrow \)

Use it! Lose it! Don't write any code for these... Do try the examples + brainstorm

Brainstorm algorithms for these problems -- what helper functions?? might help for each?
should return the jotto score for any strings s1 and s2

\[
\text{jscore}(s1, s2)
\]

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>jscore(s1, s2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>'robot'</td>
<td>'otter'</td>
<td>3</td>
</tr>
<tr>
<td>'geese'</td>
<td>'seems'</td>
<td>3</td>
</tr>
<tr>
<td>'fluff'</td>
<td>'lulls'</td>
<td>2</td>
</tr>
<tr>
<td>'pears'</td>
<td>'diner'</td>
<td>2</td>
</tr>
<tr>
<td>'xylyl'</td>
<td>'slyly'</td>
<td>4</td>
</tr>
</tbody>
</table>

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

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

\[
\text{exact_change}(t, L)
\]

<table>
<thead>
<tr>
<th>t, L</th>
<th>exact_change(t, L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>42, [25, 30, 2, 5]</td>
<td>False</td>
</tr>
<tr>
<td>42, [22, 16, 3, 2, 17]</td>
<td>True</td>
</tr>
<tr>
<td>42, [18, 21, 22]</td>
<td>False</td>
</tr>
<tr>
<td>42, [40, 17, 1]</td>
<td>False</td>
</tr>
<tr>
<td>20, [16, 3, 2, 17]</td>
<td>True</td>
</tr>
</tbody>
</table>

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

\[
\text{sort}(L)
\]

\[
\text{sort}([42, 5, 7]) \rightarrow [5, 7, 42]
\]

\[
\text{sort}([42, 7]) \rightarrow [7, 42]
\]

\[
\text{sort}([42]) \rightarrow [42]
\]

\[
\text{sort}([ ]) \rightarrow [ ]
\]

\[
\text{blsort}([1, 0, 1]) \rightarrow [0, 1, 1]
\]

\[
\text{sort}([ ]) \rightarrow [ ]
\]

should return the Longest Common Subsequence of strings S and T

\[
\text{LCS}(S, T)
\]

\[
\text{LCS}('ctga', 'tagca') \rightarrow 'tga'
\]

\[
\text{LCS}('tga', 'taacg') \rightarrow 'ta' (or 'tg')
\]

\[
\text{LCS}('tga', 'a') \rightarrow 'a'
\]

\[
\text{LCS}('gattaca', 'ctctgcgat') \rightarrow 'tta\text{c}a'
\]

4 chars

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

Use it! Lose it! Use it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it! Lose it!


should return the jotto score for any strings \( s_1 \) and \( s_2 \)

**\( \text{jscore}(s_1,s_2) \)**

\[
\begin{align*}
\text{jscore}('robot', 'otter') & \rightarrow 3 \\
\text{jscore}('geese', 'seems') & \rightarrow 3 \\
\text{jscore}('fluff', 'lulls') & \rightarrow 2 \\
\text{jscore}('pears', 'diner') & \rightarrow 2 \\
\text{jscore}('xyyl', 'slyly') & \rightarrow 4
\end{align*}
\]

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

**\( \text{sort}(L) \)**

\[
\begin{align*}
\text{sort}([42,5,7]) & \rightarrow [5,7,42] \\
\text{sort}([42,7]) & \rightarrow [7,42] \\
\text{sort}([42]) & \rightarrow [42] \\
\text{sort}([]) & \rightarrow [] \\
\text{blsort}([1,0,1]) & \rightarrow [0,1,1]
\end{align*}
\]

\( \text{binary-list sort: same as sort, but all of the #s are 0 or 1} \)

**\( \text{LCS}(S,T) \)**

\[
\begin{align*}
\text{LCS}('ctga', 'tagca') & \rightarrow 'tga' \\
\text{LCS}('tga', 'taacg') & \rightarrow 'ta' (or 'tg') \\
\text{LCS}('tga', 'a') & \rightarrow 'a' \\
\text{LCS}('gattaca', 'ctctgccat') & \rightarrow 'ttca' (4 chars)
\end{align*}
\]

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

**exact_change\( (t,L) \)**

\[
\begin{align*}
\text{exact_change}(42, [25,30,2,5]) & \rightarrow \text{False} \\
\text{exact_change}(42, [22,16,3,2,17]) & \rightarrow \text{True} \\
\text{exact_change}(42, [18,21,22]) & \rightarrow \text{False} \\
\text{exact_change}(42, [40,17,1,7]) & \rightarrow \text{False} \\
\text{exact_change}(20, [16,3,2,17]) & \rightarrow \text{True}
\end{align*}
\]

returns True if any subset of elements in \( L \) add up to \( t \); returns False otherwise
decipher('Weet bksa ed xecumeha 4!')
decipher( 'Weet bksa ed xecumeha 4!' )

Good luck on homework 4!

kxn rkfo k qbokd goouoxn ...

and have a great weekend ...